# Tarantool Documentation

*Выпуск 3.0.0*

апр. 26, 2024

# Оглавление


**1 Overview** **1**
- 1.1 What is Tarantool? . . . 1
- 1.2 Editions . . . 2
- 1.3 Use cases . . . 2

**2 Руководство для начинающих** **4**

**3 Примеры и руководства** **5**
- 3.1 Создаем свою первую базу данных на Tarantool . . . 5
- 3.2 Defining and manipulating data . . . 9
- 3.3 Replication tutorials . . . 24
- 3.4 Creating a sharded cluster . . . 52
- 3.5 Getting started with Tarantool Cluster Manager . . . 64
- 3.6 Connecting to a database using net.box . . . 79
- 3.7 Подключаемся к базе из разных языков программирования . . . 84
- 3.8 Developing applications with Tarantool . . . 103
- 3.9 SQL guides . . . 161
- 3.10 More guides . . . 193

**4 Concepts** **202**
- 4.1 Data model . . . 202
- 4.2 Fibers and cooperative multitasking . . . 202
- 4.3 Transactions . . . 202
- 4.4 Application server . . . 203
- 4.5 Sharding . . . 203
- 4.6 Triggers . . . 203
- 4.7 Replication . . . 203
- 4.8 Storage engines . . . 203

**5 Вложенный модуль box.space** **323**
- 5.1 box.schema.space.create() . . . 324
- 5.2 space_object:alter() . . . 328
- 5.3 space_object:auto_increment() . . . 329
- 5.4 space_object:bsize() . . . 329
- 5.5 space_object:count() . . . 330
- 5.6 space_object:create_index() . . . 330
- 5.7 space_object:delete() . . . 342

5.8 space_object:drop() . . . 343
5.9 space_object:format() . . . 343
5.10 space_object:frommap() . . . 347
5.11 space_object:get() . . . 348
5.12 space_object:insert() . . . 349
5.13 space_object:len() . . . 349
5.14 space_object:on_replace() . . . 350
5.15 space_object:before_replace() . . . 352
5.16 space_object:pairs() . . . 353
5.17 space_object:put() . . . 355
5.18 space_object:rename() . . . 355
5.19 space_object:replace() / put() . . . 355
5.20 space_object:run_triggers() . . . 356
5.21 space_object:select() . . . 357
5.22 space_object:truncate() . . . 359
5.23 space_object:update() . . . 360
5.24 space_object:upsert() . . . 362
5.25 space_object extensions . . . 363
5.26 box.space.create_check_constraint() . . . 363
5.27 space_object:enabled . . . 365
5.28 space_object:field_count . . . 365
5.29 space_object.id . . . 366
5.30 space_object.index . . . 366
5.31 box.space._cluster . . . 367
5.32 box.space._func . . . 367
5.33 box.space._index . . . 367
5.34 box.space._vindex . . . 368
5.35 box.space._priv . . . 368
5.36 box.space._vpriv . . . 368
5.37 box.space._schema . . . 368
5.38 box.space._sequence . . . 369
5.39 box.space._sequence_data . . . 369
5.40 box.space._space . . . 369
5.41 box.space._vspace . . . 371
5.42 box.space._space_sequence . . . 371
5.43 box.space._vspace_sequence . . . 372
5.44 box.space._user . . . 372
5.45 box.space._vuser . . . 374
5.46 box.space._ck_constraint . . . 374
5.47 box.space._collation . . . 374
5.48 box.space._vcollation . . . 375
5.49 Представления системных спейсов . . . 375
5.50 box.space._session_settings . . . 376

**6 Администрирование 378**
6.1 Application environment . . . 378
6.2 Starting and stopping instances . . . 382
6.3 Managing modules . . . 387
6.4 Журналирование . . . 391
6.5 Безопасность . . . 392
6.6 Управление доступом . . . 394
6.7 Шардирование с vshard . . . 409
6.8 Replication administration . . . 418
6.9 Просмотр состояния сервера . . . 427

6.10 Контроль за фоновыми программами . . . 438
6.11 Аварийное восстановление . . . 442
6.12 Резервное копирование . . . 444
6.13 Обновление . . . 445
6.14 Замечания по поводу некоторых операционных систем . . . 461
6.15 Сообщения об ошибках . . . 462
6.16 Руководство по разрешению проблем . . . 462
6.17 Monitoring . . . 473

**7 Коннекторы** **527**
7.1 C . . . 527
7.2 Go . . . 532
7.3 Java . . . 533
7.4 Python . . . 534
7.5 Community-supported connectors . . . 535
7.6 Протокол . . . 551
7.7 Пример пакета данных . . . 551
7.8 Настройка окружения для работы с примерами . . . 552
7.9 Интерпретация возвращаемых значений . . . 552

**8 Tarantool Enterprise Edition** **554**
8.1 Настройка . . . 555
8.2 Инструкции по повышению безопасности . . . 557
8.3 Аудит безопасности . . . 560
8.4 Tuple compression . . . 563
8.5 WAL extensions . . . 573
8.6 Read views . . . 574
8.7 Flight recorder . . . 582
8.8 Журнал аудита . . . 584
8.9 Upgrading space schema . . . 592
8.10 Метрики системы мониторинга . . . 600
8.11 Устаревшие функции . . . 602
8.12 Modules . . . 617

**9 Справочники** **619**
9.1 Configuration reference . . . 619
9.2 Configuration reference (box.cfg) . . . 683
9.3 Tooling . . . 736
9.4 SQL reference . . . 882
9.5 Справочник по встроенным модулям . . . 989
9.6 Справочник по сторонним библиотекам . . . 1399
9.7 Справочник по C API . . . 1467
9.8 Детали реализации . . . 1496
9.9 Ограничения . . . 1537
9.10 Рекомендации по Lua-синтаксису . . . 1539

**10 Contributing** **1541**
10.1 How to get involved in Tarantool . . . 1541
10.2 How to write release notes . . . 1546
10.3 Building to contribute . . . 1546
10.4 Добавление собственного модуля . . . 1552
10.5 Рекомендации . . . 1552

**11 Примечания к версиям** **1629**
11.1 Supported versions . . . 1629

ГЛАВА 1

# Overview

## 1.1 What is Tarantool?

Tarantool combines an in-memory DBMS and a Lua server in a single platform providing ACID-compliant storage. It comes in two *editions*: Community and Enterprise. The *use cases* for Tarantool vary from ultra-fast cache to product data marts and smart queue services.

Here are some of Tarantool's key characteristics:

- **Easy handling of OLTP workloads**: processes hundreds of thousands RPS
- **Data integrity**: *write-ahead log (WAL)* and *data snapshots*
- **Cooperative multitasking**: transactions are performed in *lightweight coroutines* with no interthread locking
- **Advanced indexing**: *composite indexes*, *locale support*, indexing by *nested fields and arrays*
- **Compute close to data**: *Lua server* and Just-In-Time compiler on board
- **Durable distributed storage**: multiple failover modes and *RAFT-based synchronous replication* available

Tarantool allows executing code alongside data, which helps increase the speed of operations. Developers can *implement any business logic with Lua*, and a single Tarantool instance can also receive *SQL requests*.

Tarantool has a variety of compatible modules (Lua rocks). You can pick the ones that you need and install them manually.

Tarantool runs on Linux (x86_64, aarch64), Mac OS X (x86_64, M1), and FreeBSD (x86_64).

You can use Tarantool with a programming language you're familiar with. For this purpose, a number of *connectors* are provided.

## 1.2 Editions

Tarantool comes in two editions: the open-source **Community Edition (CE)** and the commercial **Enterprise Edition (EE)**.

**Tarantool Community Edition** lets you develop applications and speed up a system in operation. It features *synchronous replication*, affords easy *scalability*, and includes tools to develop efficient *applications*. The Tarantool community helps with any practical questions regarding the Community Edition.

**Tarantool Enterprise Edition** provides advanced tools for administration, deployment, and security management, along with premium support services. This edition includes all the Community Edition features and is more predictable in terms of solution cost and maintenance. The Enterprise Edition is shipped as an SDK and includes a number of closed-source modules. See the *Tarantool Enterprise Edition* documentation.

## 1.3 Use cases

### 1.3.1 Fast first-class storage

- Primary storage
  - No secondary storage required
- Tolerance to high write loads
- Support of relational approaches
- Composite secondary indexes
  - Data access, data slices
- Predictable request latency

### 1.3.2 Advanced cache

- Write-behind caching
- Secondary index support
- Complex invalidation algorithm support

### 1.3.3 Smart queue

- Support of various identification techniques
- Advanced task lifecycle management
  - Task scheduling
  - Archiving of completed tasks

### 1.3.4 Data-centric applications

- Arbitrary data flows from many sources
- Incoming data processing
- Storage
- Background cycle processing
  - Scheduling support

# ГЛАВА 2

## Руководство для начинающих

The *Creating your first Tarantool database* section will get you acquainted with Tarantool. We will start Tarantool, create a data schema, and write our first data. You'll get an understanding of the technology and learn about the basic terms and features.

To continue exploring Tarantool and its ecosystem, you might want to check out Tarantool *tutorials and guides*.

# ГЛАВА 3

# Примеры и руководства

Глава содержит полезные примеры использования Tarantool, а также практические руководства для тех, кто хочет получше познакомиться с платформой.

Если вы еще не использовали Tarantool, пожалуйста, сначала ознакомьтесь с *Руководством для начинающих*.

## 3.1 Создаем свою первую базу данных на Tarantool

**Example on GitHub**: create_db

In this tutorial, you create a Tarantool database, write data to it, and select data from this database.

### 3.1.1 Prerequisites

Before starting this tutorial:

- Install the *tt* utility.
- Install Tarantool.

---

**Примечание:** The tt utility provides the ability to install Tarantool software using the *tt install* command.

---

### 3.1.2 Creating an application

The *tt create* command can be used to *create an application* from a predefined or custom template. In this tutorial, the application layout is prepared manually:

1. Create a tt environment in the current directory using the *tt init* command.
2. Inside the `instances.enabled` directory of the created tt environment, create the `create_db` directory.
3. Inside `instances.enabled/create_db`, create the `instances.yml` and `config.yaml` files:
   - `instances.yml` specifies instances to run in the current environment. In this example, there is one instance:

     ```
     instance001:
     ```

   - `config.yaml` contains basic instance *configuration*:

     ```
     groups:
       group001:
         replicasets:
           replicaset001:
             instances:
               instance001:
                 iproto:
                   listen:
                   - uri: '127.0.0.1:3301'
     ```

     The instance in the configuration accepts incoming requests on the 3301 port.

### 3.1.3 Working with the database

#### Starting an instance

1. Start the Tarantool instance from the tt environment directory using *tt start*:

   ```
   $ tt start create_db
   ```

2. To check the running instance, use the *tt status* command:

   ```
   $ tt status create_db
   INSTANCE                        STATUS      PID
   create_db:instance001           RUNNING     54560
   ```

3. Connect to the instance with *tt connect*:

   ```
   $ tt connect create_db:instance001
      • Connecting to the instance...
      • Connected to create_db:instance001

   create_db:instance001>
   ```

   This command opens an interactive Tarantool console with the `create_db:instance001>` prompt. Now you can enter requests in the command line.

### Creating a space

1. Create a *space* named `bands`:

```
create_db:instance001> box.schema.space.create('bands')
---
- engine: memtx
  before_replace: 'function: 0x010229d788'
  field_count: 0
  is_sync: false
  is_local: false
  on_replace: 'function: 0x010229d750'
  temporary: false
  index: []
  type: normal
  enabled: false
  name: bands
  id: 512
- created
...
```

2. Форматируйте созданный спейс, указывая имена и *типы* полей:

```
create_db:instance001> box.space.bands:format({
                           { name = 'id', type = 'unsigned' },
                           { name = 'band_name', type = 'string' },
                           { name = 'year', type = 'unsigned' }
                       })
---
...
```

### Creating indexes

1. Create the primary *index* based on the `id` field:

```
create_db:instance001> box.space.bands:create_index('primary', { parts = { 'id' } })
---
- unique: true
  parts:
  - fieldno: 1
    sort_order: asc
    type: unsigned
    exclude_null: false
    is_nullable: false
  hint: true
  id: 0
  type: TREE
  space_id: 512
  name: primary
...
```

2. Create the secondary index based on the `band_name` field:

```
create_db:instance001> box.space.bands:create_index('secondary', { parts = { 'band_
↪name' } })
---
- unique: true
  parts:
  - fieldno: 2
    sort_order: asc
    type: string
    exclude_null: false
    is_nullable: false
  hint: true
  id: 1
  type: TREE
  space_id: 512
  name: secondary
...
```

### Writing and selecting data

1. Insert three *tuples* into the space:

```
create_db:instance001> box.space.bands:insert { 1, 'Roxette', 1986 }
---
- [1, 'Roxette', 1986]
...
create_db:instance001> box.space.bands:insert { 2, 'Scorpions', 1965 }
---
- [2, 'Scorpions', 1965]
...
create_db:instance001> box.space.bands:insert { 3, 'Ace of Base', 1987 }
---
- [3, 'Ace of Base', 1987]
...
```

2. Select a tuple using the `primary` index:

```
create_db:instance001> box.space.bands:select { 3 }
---
- - [3, 'Ace of Base', 1987]
...
```

3. Select tuples using the `secondary` index:

```
create_db:instance001> box.space.bands.index.secondary:select{'Scorpions'}
---
- - [2, 'Scorpions', 1965]
...
```

## 3.2 Defining and manipulating data

This section contains guides on performing data operations in Tarantool.

### 3.2.1 Примеры CRUD-операций

#### Using data operations

This section shows basic usage scenarios and typical errors for each *data operation* in Tarantool: *INSERT*, *DELETE*, *UPDATE*, *UPSERT*, *REPLACE*, and *SELECT*. Before trying out the examples, you need to bootstrap a Tarantool instance as shown below.

```
-- Create a space --
bands = box.schema.space.create('bands')

-- Specify field names and types --
box.space.bands:format({
    { name = 'id', type = 'unsigned' },
    { name = 'band_name', type = 'string' },
    { name = 'year', type = 'unsigned' }
})

-- Create a primary index --
box.space.bands:create_index('primary', { parts = { 'id' } })

-- Create a unique secondary index --
box.space.bands:create_index('band', { parts = { 'band_name' } })

-- Create a non-unique secondary index --
box.space.bands:create_index('year', { parts = { { 'year' } }, unique = false })

-- Create a multi-part index --
box.space.bands:create_index('year_band', { parts = { { 'year' }, { 'band_name' } } })
```

#### INSERT

The *space_ object.insert* method accepts a well-formatted tuple.

```
-- Insert a tuple with a unique primary key --
tarantool> bands:insert{1, 'Scorpions', 1965}
---
- [1, 'Scorpions', 1965]
...
```

`insert` also checks all the keys for duplicates.

```
-- Try to insert a tuple with a duplicate primary key --
tarantool> bands:insert{1, 'Scorpions', 1965}
---
- error: Duplicate key exists in unique index "primary" in space "bands" with old
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    tuple - [1, "Scorpions", 1965] and new tuple - [1, "Scorpions", 1965]
...

-- Try to insert a tuple with a duplicate secondary key --
tarantool> bands:insert{2, 'Scorpions', 1965}
---
- error: Duplicate key exists in unique index "band" in space "bands" with old tuple
    - [1, "Scorpions", 1965] and new tuple - [2, "Scorpions", 1965]
...

-- Insert a second tuple with unique primary and secondary keys --
tarantool> bands:insert{2, 'Pink Floyd', 1965}
---
- [2, 'Pink Floyd', 1965]
...

-- Delete all tuples --
tarantool> bands:truncate()
---
...
```

### DELETE

*space_object.delete* allows you to delete a tuple identified by the primary key.

```
-- Insert test data --
tarantool> bands:insert{1, 'Roxette', 1986}
           bands:insert{2, 'Scorpions', 1965}
           bands:insert{3, 'Ace of Base', 1987}
           bands:insert{4, 'The Beatles', 1960}

-- Delete a tuple with an existing key --
tarantool> bands:delete{4}
---
- [4, 'The Beatles', 1960]
...
tarantool> bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'Ace of Base', 1987]
...
```

You can also use *index_object.delete* to delete a tuple by the specified unique index.

```
-- Delete a tuple by the primary index --
tarantool> bands.index.primary:delete{3}
---
- [3, 'Ace of Base', 1987]
...
tarantool> bands:select()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
...

-- Delete a tuple by a unique secondary index --
tarantool> bands.index.band:delete{'Scorpions'}
---
- [2, 'Scorpions', 1965]
...
tarantool> bands:select()
---
- - [1, 'Roxette', 1986]
...

-- Try to delete a tuple by a non-unique secondary index --
tarantool> bands.index.year:delete(1986)
---
- error: Get() doesn't support partial keys and non-unique indexes
...
tarantool> bands:select()
---
- - [1, 'Roxette', 1986]
...

-- Try to delete a tuple by a partial key --
tarantool> bands.index.year_band:delete('Roxette')
---
- error: Invalid key part count in an exact match (expected 2, got 1)
...

-- Delete a tuple by a full key --
tarantool> bands.index.year_band:delete{1986, 'Roxette'}
---
- [1, 'Roxette', 1986]
...
tarantool> bands:select()
---
- []
...

-- Delete all tuples --
tarantool> bands:truncate()
---
...
```

### UPDATE

*space_object.update* allows you to update a tuple identified by the primary key. Similarly to `delete`, the `update` method accepts a full key and also an operation to execute.

```
-- Insert test data --
tarantool> bands:insert{1, 'Roxette', 1986}
           bands:insert{2, 'Scorpions', 1965}
           bands:insert{3, 'Ace of Base', 1987}
           bands:insert{4, 'The Beatles', 1960}

-- Update a tuple with an existing key --
tarantool> bands:update({2}, {{'=', 2, 'Pink Floyd'}})
---
- [2, 'Pink Floyd', 1965]
...

tarantool> bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'Pink Floyd', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
...
```

*index_object.update* updates a tuple identified by the specified unique index.

```
-- Update a tuple by the primary index --
tarantool> bands.index.primary:update({2}, {{'=', 2, 'The Rolling Stones'}})
---
- [2, 'The Rolling Stones', 1965]
...

tarantool> bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'The Rolling Stones', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
...

-- Update a tuple by a unique secondary index --
tarantool> bands.index.band:update({'The Rolling Stones'}, {{'=', 2, 'The Doors'}})
---
- [2, 'The Doors', 1965]
...

tarantool> bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'The Doors', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...

-- Try to update a tuple by a non-unique secondary index --
tarantool> bands.index.year:update({1965}, {{'=', 2, 'Scorpions'}})
---
- error: Get() doesn't support partial keys and non-unique indexes
...
tarantool> bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'The Doors', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
...

-- Delete all tuples --
tarantool> bands:truncate()
---
...
```

### UPSERT

*space_object.upsert* updates an existing tuple or inserts a new one:

- If the existing tuple is found by the primary key, Tarantool applies the update operation to this tuple and ignores the new tuple.
- If no existing tuple is found, Tarantool inserts the new tuple and ignores the update operation.

```
tarantool> bands:insert{1, 'Scorpions', 1965}
---
- [1, 'Scorpions', 1965]
...
-- As the first argument, upsert accepts a tuple, not a key --
tarantool> bands:upsert({2}, {{'=', 2, 'Pink Floyd'}})
---
- error: Tuple field 2 (band_name) required by space format is missing
...
tarantool> bands:select()
---
- - [1, 'Scorpions', 1965]
...
tarantool> bands:delete(1)
---
- [1, 'Scorpions', 1965]
...
```

`upsert` acts as `insert` when no existing tuple is found by the primary key.

```
tarantool> bands:upsert({1, 'Scorpions', 1965}, {{'=', 2, 'The Doors'}})
---
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- As you can see, {1, 'Scorpions', 1965} is inserted, --
-- and the update operation is not applied. --
tarantool> bands:select()
---
- - [1, 'Scorpions', 1965]
...

-- upsert with the same primary key but different values in other fields --
-- applies the update operation and ignores the new tuple. --
tarantool> bands:upsert({1, 'Scorpions', 1965}, {{'=', 2, 'The Doors'}})
---
...
tarantool> bands:select()
---
- - [1, 'The Doors', 1965]
...
```

upsert searches for the existing tuple by the primary index, not by the secondary index. This can lead to a duplication error if the tuple violates a secondary index uniqueness.

```
tarantool> bands:upsert({2, 'The Doors', 1965}, {{'=', 2, 'Pink Floyd'}})
---
- error: Duplicate key exists in unique index "band" in space "bands" with old tuple
    - [1, "The Doors", 1965] and new tuple - [2, "The Doors", 1965]
...
tarantool> bands:select()
---
- - [1, 'The Doors', 1965]
...

-- This works if uniqueness is preserved. --
tarantool> bands:upsert({2, 'The Beatles', 1960}, {{'=', 2, 'Pink Floyd'}})
---
...
tarantool> bands:select()
---
- - [1, 'The Doors', 1965]
  - [2, 'The Beatles', 1960]
...

-- Delete all tuples --
tarantool> bands:truncate()
---
...
```

### REPLACE

*space_object.replace* accepts a well-formatted tuple and searches for the existing tuple by the primary key of the new tuple:

- If the existing tuple is found, Tarantool deletes it and inserts the new tuple.
- If no existing tuple is found, Tarantool inserts the new tuple.

```
tarantool> bands:replace{1, 'Scorpions', 1965}
---
- [1, 'Scorpions', 1965]
...
tarantool> bands:select()
---
- - [1, 'Scorpions', 1965]
...
tarantool> bands:replace{1, 'The Beatles', 1960}
---
- [1, 'The Beatles', 1960]
...
tarantool> bands:select()
---
- - [1, 'The Beatles', 1960]
...
tarantool> bands:truncate()
---
...
```

`replace` can violate unique constraints, like `upsert` does.

```
tarantool> bands:insert{1, 'Scorpions', 1965}
- [1, 'Scorpions', 1965]
...
tarantool> bands:insert{2, 'The Beatles', 1960}
---
- [2, 'The Beatles', 1960]
...
tarantool> bands:replace{2, 'Scorpions', 1965}
---
- error: Duplicate key exists in unique index "band" in space "bands" with old tuple
    - [1, "Scorpions", 1965] and new tuple - [2, "Scorpions", 1965]
...
tarantool> bands:truncate()
---
...
```

### SELECT

The *space_object.select* request searches for a tuple or a set of tuples in the given space by the primary key. To search by the specified index, use *index_object.select*. These methods work with any keys, including unique and non-unique, full and partial. If a key is partial, `select` searches by all keys where the prefix matches the specified key part.

```
tarantool> bands:insert{1, 'Roxette', 1986}
           bands:insert{2, 'Scorpions', 1965}
           bands:insert{3, 'The Doors', 1965}
           bands:insert{4, 'The Beatles', 1960}

tarantool> bands:select(1)
---
- - [1, 'Roxette', 1986]
...

tarantool> bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'The Doors', 1965]
  - [4, 'The Beatles', 1960]
...

tarantool> bands.index.primary:select(2)
---
- - [2, 'Scorpions', 1965]
...

tarantool> bands.index.band:select('The Doors')
---
- - [3, 'The Doors', 1965]
...

tarantool> bands.index.year:select(1965)
---
- - [2, 'Scorpions', 1965]
  - [3, 'The Doors', 1965]
...
```

#### Using box.space functions to read _space tuples

This example illustrates how to look at all the spaces, and for each display: approximately how many tuples it contains, and the first field of its first tuple. The function uses the Tarantool's `box.space` functions `len()` and `pairs()`. The iteration through the spaces is coded as a scan of the `_space` system space, which contains metadata. The third field in `_space` contains the space name, so the key instruction `space_name = v[3]` means `space_name` is the `space_name` field in the tuple of `_space` that we've just fetched with `pairs()`. The function returns a table:

```
function example()
  local tuple_count, space_name, line
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  local ta = {}
  for k, v in box.space._space:pairs() do
    space_name = v[3]
    if box.space[space_name].index[0] ~= nil then
      tuple_count = '1 or more'
    else
      tuple_count = '0'
    end
    line = space_name .. ' tuple_count =' .. tuple_count
    if tuple_count == '1 or more' then
      for k1, v1 in box.space[space_name]:pairs() do
        line = line .. '. first field in first tuple = ' .. v1[1]
        break
      end
    end
    table.insert(ta, line)
  end
  return ta
end
```

The output below shows what happens if you invoke this function:

```
tarantool> example()
---
- - _schema tuple_count =1 or more. first field in first tuple = cluster
  - _space tuple_count =1 or more. first field in first tuple = 272
  - _vspace tuple_count =1 or more. first field in first tuple = 272
  - _index tuple_count =1 or more. first field in first tuple = 272
  - _vindex tuple_count =1 or more. first field in first tuple = 272
  - _func tuple_count =1 or more. first field in first tuple = 1
  - _vfunc tuple_count =1 or more. first field in first tuple = 1
  - _user tuple_count =1 or more. first field in first tuple = 0
  - _vuser tuple_count =1 or more. first field in first tuple = 0
  - _priv tuple_count =1 or more. first field in first tuple = 1
  - _vpriv tuple_count =1 or more. first field in first tuple = 1
  - _cluster tuple_count =1 or more. first field in first tuple = 1
...
```

### Using box.space functions to organize a _space tuple

This examples shows how to display field names and field types of a system space – using metadata to find metadata.

Для начала: как можно сделать выборку кортежа из `_space`, который описывает `_space`?

A simple way is to look at the constants in `box.schema`, which shows that there is an item named SPACE_ID == 288, so these statements retrieve the correct tuple:

```
box.space._space:select{ 288 }
-- или --
box.space._space:select{ box.schema.SPACE_ID }
```

Another way is to look at the tuples in box.space._index, which shows that there is a secondary index named „name“ for a space number 288, so this statement also retrieve the correct tuple:

```
box.space._space.index.name:select{ '_space' }
```

Однако непросто прочитать информацию из полученного кортежа:

```
tarantool> box.space._space.index.name:select{'_space'}
---
- - [280, 1, '_space', 'memtx', 0, {}, [{'name': 'id', 'type': 'num'}, {'name': 'owner',
        'type': 'num'}, {'name': 'name', 'type': 'str'}, {'name': 'engine', 'type': 'str
↪'},
      {'name': 'field_count', 'type': 'num'}, {'name': 'flags', 'type': 'str'}, {
        'name': 'format', 'type': '*'}]]
...
```

Информация подается бессистемно, поскольку по формату поле №7 содержит рекомендованные имена и типы данных. Как же получить эти данные? Поскольку очевидно, что поле №7 представляет собой ассоциативный массив, цикл for проведет организацию данных:

```
tarantool> do
         >   local tuple_of_space = box.space._space.index.name:get{'_space'}
         >   for _, field in ipairs(tuple_of_space[7]) do
         >     print(field.name .. ', ' .. field.type)
         >   end
         > end
id, num
owner, num
name, str
engine, str
field_count, num
flags, str
format, *
---
...
```

### 3.2.2 Using indexes

#### Создание индекса

Перед тем, как вставлять кортежи в спейс или выбирать из него кортежи, нужно обязательно создать индекс для этого спейса.

Простая операция по *созданию индекса* выглядит следующим образом:

box.space.space-name:create_index('*index-name*')

При этом создается уникальный *TREE-индекс* по первому полю всех кортежей (обычно его называют «Field#1»). Предполагается, что индексируемое поле является числовым.

Рекомендуется проектировать модель данных так, чтобы первичные ключи были первыми полями в кортеже. Это позволяет быстрее сравнивать кортежи, учитывая специфику хранения данных и способ организации сравнений в Tarantool.

Вот простой запрос *SELECT*:

`box.space.space-name:select(value)`

Такой запрос ищет отдельный кортеж, используя первичный индекс. Поскольку первичный индекс всегда уникален, то данный запрос вернет не более одного кортежа. Можно также вызвать `select()` без аргументов, чтобы вернуть все кортежи. Осторожно! Если вызвать `select()` без аргументов в огромном спейсе, ваш экземпляр зависнет.

An index definition may also include identifiers of tuple fields and their expected **types**. See allowed indexed field types in section *Details about indexed field types*:

```
box.space.space-name:create_index(index-name, {type = 'tree', parts = {{field = 1, type␣
↪= 'unsigned'}}}
```

Определения спейса и определения индексов хранятся в системных спейсах Tarantool *_space* и *_index* соответственно.

---

**Примечание**

Полную информацию о создании индексов, например о создании индексов по массиву (multikey), индексов с использованием пути `path` или функциональных индексов см. в справочнике *space_object:create_index()*.

---

### Операции с индексами

Операции с индексами производятся автоматически. Если запрос на изменение данных меняет данные в кортеже, то изменятся и ключи индекса для данного кортежа.

1. Для примера создадим спейс с именем `bands`:

```
bands = box.schema.space.create('bands')
```

2. Отформатируем созданный спейс, указав имена и типы полей:

```
box.space.bands:format({
    { name = 'id', type = 'unsigned' },
    { name = 'band_name', type = 'string' },
    { name = 'year', type = 'unsigned' }
})
```

3. Создадим **первичный** индекс с именем `primary`:

```
box.space.bands:create_index('primary', { parts = { 'id' } })
```

   This index is based on the `id` field of each tuple.

4. Вставим несколько *кортежей* в спейс:

```
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.space.bands:insert { 5, 'Pink Floyd', 1965 }
box.space.bands:insert { 6, 'The Rolling Stones', 1962 }
box.space.bands:insert { 7, 'The Doors', 1965 }
box.space.bands:insert { 8, 'Nirvana', 1987 }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
box.space.bands:insert { 9, 'Led Zeppelin', 1968 }
box.space.bands:insert { 10, 'Queen', 1970 }
```

5. Create **secondary indexes**:

```
-- Create a unique secondary index --
box.space.bands:create_index('band', { parts = { 'band_name' } })

-- Create a non-unique secondary index --
box.space.bands:create_index('year', { parts = { { 'year' } }, unique = false })
```

6. Create a **multi-part index** with two parts:

```
box.space.bands:create_index('year_band', { parts = { { 'year' }, { 'band_name' } }␣
↪})
```

Можно использовать такие варианты SELECT:

- Помимо условия равенства, при поиске могут использоваться и другие **условия сравнения**:

```
-- Select maximum 3 tuples with the key value greater than 1965 --
select_greater = bands.index.year:select({ 1965 }, { iterator = 'GT', limit = 3 })
--[[
---
- - [9, 'Led Zeppelin', 1968]
  - [10, 'Queen', 1970]
  - [1, 'Roxette', 1986]
...
--]]
```

  Есть такие *операторы сравнения*:

  - `LT` — «меньше»
  - `LE` — «меньше или равно»
  - `GT` — «больше»
  - `GE` for «greater than or equal»
  - `EQ` for «equal»
  - `REQ` — «равно, результаты отсортированы в порядке убывания по ключу»

  Только для TREE-индексов есть смысл в сравнении значений. Итераторы для других типов индексов немного отличаются и работают по-другому. Более подробную информацию см. в разделе *Типы итераторов*.

  Обратите внимание, что мы не используем имя индекса — значит, здесь мы используем первичный индекс.

  Этот вариант поиска может вернуть более одного кортежа. Если используется оператор сравнения LT, LE или REQ, то кортежи будут отсортированы по ключу в порядке убывания. Во всех остальных случаях — в порядке возрастания.

- Для поиска можно использовать **вторичный индекс**.

```
-- Select a tuple by the specified secondary key value --
select_secondary = bands.index.band:select { 'The Doors' }
--[[
---
- - [7, 'The Doors', 1965]
...
--]]
```

- **Поиск по частям ключа:** Можно производить поиск по частям ключа, начиная с префикса ключа. Обратите внимание, что поиск по части ключа доступен только в TREE-индексах.

```
-- Select tuples by the specified partial key value --
select_multipart_partial = bands.index.year_band:select { 1965 }
--[[
---
- - [5, 'Pink Floyd', 1965]
  - [2, 'Scorpions', 1965]
  - [7, 'The Doors', 1965]
...
--]]
```

- Поиск можно производить по всем полям через запись в виде таблицы:

```
-- Select a tuple by the specified multi-part secondary key value --
select_multipart = bands.index.year_band:select { 1960, 'The Beatles' }
--[[
---
- - [4, 'The Beatles', 1960]
...
--]]
```

**Примечание**

С некоторыми ограничениями можно добавлять, удалять или изменять определения во время исполнения кода. Более подробную информацию об операциях с индексами см. в справочнике по *вложенному модулю box.index*.

### 3.2.3 Using sequences

A **sequence** is a generator of ordered integer values.

As with spaces and indexes, you should specify the sequence **name** and let Tarantool generate a unique numeric identifier (sequence ID).

As well, you can specify several options when creating a new sequence. The options determine the values that are generated whenever the sequence is used.

### Options for box.schema.sequence.create()

| Option name | Type and meaning | Default | Examples |
|---|---|---|---|
| `start` | Integer. The value to generate the first time a sequence is used | 1 | `start=0` |
| `min` | Integer. Values smaller than this cannot be generated | 1 | `min=-1000` |
| `max` | Integer. Values larger than this cannot be generated | 9223372036854775807 | `max=0` |
| `cycle` | Boolean. Whether to start again when values cannot be generated | false | `cycle=true` |
| `cache` | Integer. The number of values to store in a cache | 0 | `cache=0` |
| `step` | Integer. What to add to the previous generated value, when generating a new value | 1 | `step=-1` |
| `if_not_exists` | Boolean. If this is true and a sequence with this name exists already, ignore other options and use the existing values | `false` | `if_not_exists=true` |

Once a sequence exists, it can be altered, dropped, reset, forced to generate the next value, or associated with an index.

### Associating a sequence with an index

First, create a sequence:

```
-- Create a sequence --
box.schema.sequence.create('id_seq',{min=1000, start=1000})
--[[
---
- step: 1
  id: 1
  min: 1000
  cache: 0
  uid: 1
  cycle: false
  name: id_seq
  start: 1000
  max: 9223372036854775807
...
--]]
```

The result shows that the new sequence has all default values, except for the two that were specified, `min` and `start`.

Get the next value from the sequence by calling the `next()` function:

```
-- Get the next item --
box.sequence.id_seq:next()
--[[
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- 1000
...
--]]
```

The result is the same as the start value. The next call increases the value by one (the default sequence step).

Create a space and specify that its primary key should be generated from the sequence:

```
-- Create a space --
box.schema.space.create('customers')

-- Create an index that uses the sequence --
box.space.customers:create_index('primary',{ sequence = 'id_seq' })
--[[
---
- parts:
  - type: unsigned
    is_nullable: false
    fieldno: 1
  sequence_id: 1
  id: 0
  space_id: 513
  unique: true
  hint: true
  type: TREE
  name: primary
  sequence_fieldno: 1
...
--]]
```

Insert a tuple without specifying a value for the primary key:

```
-- Insert a tuple without the primary key value --
box.space.customers:insert{ nil, 'Adams' }
--[[
---
- [1001, 'Adams']
...
--]]
```

The result is a new tuple where the first field is assigned the next value from the sequence. This arrangement, where the system automatically generates the values for a primary key, is sometimes called «auto-incrementing» or «identity».

For syntax and implementation details, see the reference for *box.schema.sequence*.

## 3.3 Replication tutorials

### 3.3.1 Master-replica: manual failover

**Example on GitHub**: manual_leader

This tutorial shows how to configure and work with a replica set with manual failover.

#### Prerequisites

Before starting this tutorial:

1. Install the *tt* utility.
2. Create a tt environment in the current directory by executing the *tt init* command.
3. Inside the `instances.enabled` directory of the created tt environment, create the `manual_leader` directory.
4. Inside `instances.enabled/manual_leader`, create the `instances.yml` and `config.yaml` files:
   - `instances.yml` specifies instances to *run* in the current environment and should look like this:

     ```
     instance001:
     instance002:
     ```

   - The `config.yaml` file is intended to store a *replica set configuration*.

#### Configuring a replica set

This section describes how to configure a replica set in `config.yaml`.

#### Step 1: Configuring a failover mode

First, set the *replication.failover* option to `manual`:

```
replication:
  failover: manual
```

#### Step 2: Defining a replica set topology

Define a replica set topology inside the *groups* section:

- The *leader* option sets `instance001` as a replica set leader.
- The *iproto.listen* option specifies an address used to listen for incoming requests and allows replicas to communicate with each other.

```
groups:
  group001:
    replicasets:
      replicaset001:
        leader: instance001
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
```

### Step 3: Creating a user for replication

In the *credentials* section, create the `replicator` user with the `replication` role:

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]
```

### Step 4: Specifying advertise URIs

Set *iproto.advertise.peer* to advertise the current instance to other replica set members:

```
iproto:
  advertise:
    peer:
      login: replicator
```

### Resulting configuration

The resulting replica set configuration should look as follows:

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]

iproto:
  advertise:
    peer:
      login: replicator

replication:
  failover: manual

groups:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
group001:
  replicasets:
    replicaset001:
      leader: instance001
      instances:
        instance001:
          iproto:
            listen:
            - uri: '127.0.0.1:3301'
        instance002:
          iproto:
            listen:
            - uri: '127.0.0.1:3302'
```

### Working with a replica set

### Starting instances

1. After configuring a replica set, execute the *tt start* command from the *tt environment directory*:

```
$ tt start manual_leader
   • Starting an instance [manual_leader:instance001]...
   • Starting an instance [manual_leader:instance002]...
```

2. Check that instances are in the `RUNNING` status using the *tt status* command:

```
$ tt status manual_leader
INSTANCE                      STATUS      PID
manual_leader:instance001     RUNNING     15272
manual_leader:instance002     RUNNING     15273
```

### Checking a replica set status

1. Connect to `instance001` using *tt connect*:

```
$ tt connect manual_leader:instance001
   • Connecting to the instance...
   • Connected to manual_leader:instance001
```

2. Make sure that the instance is in the `running` state by executing *box.info.status*:

```
manual_leader:instance001> box.info.status
---
- running
...
```

3. Check that the instance is writable using `box.info.ro`:

```
manual_leader:instance001> box.info.ro
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- false
...
```

4. Execute `box.info.replication` to check a replica set status. For `instance002`, `upstream.status` and `downstream.status` should be `follow`.

```
manual_leader:instance001> box.info.replication
---
- 1:
    id: 1
    uuid: 9bb111c2-3ff5-36a7-00f4-2b9a573ea660
    lsn: 7
    name: instance001
  2:
    id: 2
    uuid: 4cfa6e3c-625e-b027-00a7-29b2f2182f23
    lsn: 0
    upstream:
      status: follow
      idle: 0.3893879999996
      peer: replicator@127.0.0.1:3302
      lag: 0.00028800964355469
    name: instance002
    downstream:
      status: follow
      idle: 0.37777199999982
      vclock: {1: 7}
      lag: 0
...
```

   To see the diagrams that illustrate how the `upstream` and `downstream` connections look, refer to *Monitoring a replica set*.

### Adding data

To check that a replica (`instance002`) gets all updates from the master, follow the steps below:

1. On `instance001`, create a space and add data as described in *CRUD operation examples*.
2. Open the second terminal, connect to `instance002` using `tt connect`, and use the `select` operation to make sure data is replicated.
3. Check that *box.info.vclock* values are the same on both instances:
   - `instance001`:

     ```
     manual_leader:instance001> box.info.vclock
     ---
     - {1: 21}
     ...
     ```

   - `instance002`:

```
manual_leader:instance002> box.info.vclock
---
- {1: 21}
...
```

**Примечание:** Note that a `vclock` value might include the `0` component that is related to local space operations and might differ for different instances in a replica set.

### Adding instances

This section describes how to add a new replica to a replica set.

### Adding an instance to the configuration

1. Add `instance003` to the `instances.yml` file:

```
instance001:
instance002:
instance003:
```

2. Add `instance003` with the specified `iproto.listen` option to the `config.yaml` file:

```
groups:
  group001:
    replicasets:
      replicaset001:
        leader: instance001
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
```

### Starting an instance

1. Open the third terminal to work with a new instance. Start `instance003` using `tt start`:

```
$ tt start manual_leader:instance003
   • Starting an instance [manual_leader:instance003]...
```

2. Check a replica set status using `tt status`:

```
$ tt status manual_leader
INSTANCE                        STATUS      PID
manual_leader:instance001       RUNNING     15272
manual_leader:instance002       RUNNING     15273
manual_leader:instance003       RUNNING     15551
```

### Reloading configuration

After you added `instance003` to the configuration and started it, you need to reload configurations on all instances. This is required to allow `instance001` and `instance002` to get data from the new instance in case it becomes a master.

1. Connect to `instance003` using `tt connect`:

```
$ tt connect manual_leader:instance003
   • Connecting to the instance...
   • Connected to manual_leader:instance001
```

2. Reload configurations on all three instances using the `reload()` function provided by the *config* module:

   - `instance001`:

```
manual_leader:instance001> require('config'):reload()
---
...
```

   - `instance002`:

```
manual_leader:instance002> require('config'):reload()
---
...
```

   - `instance003`:

```
manual_leader:instance003> require('config'):reload()
---
...
```

3. Execute `box.info.replication` to check a replica set status. Make sure that `upstream.status` and `downstream.status` are `follow` for `instance003`.

```
manual_leader:instance001> box.info.replication
---
- 1:
    id: 1
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    uuid: 9bb111c2-3ff5-36a7-00f4-2b9a573ea660
    lsn: 21
    name: instance001
  2:
    id: 2
    uuid: 4cfa6e3c-625e-b027-00a7-29b2f2182f23
    lsn: 0
    upstream:
      status: follow
      idle: 0.052655000000414
      peer: replicator@127.0.0.1:3302
      lag: 0.00010204315185547
    name: instance002
    downstream:
      status: follow
      idle: 0.09503500000028
      vclock: {1: 21}
      lag: 0.00026917457580566
  3:
    id: 3
    uuid: 9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6
    lsn: 0
    upstream:
      status: follow
      idle: 0.77522099999987
      peer: replicator@127.0.0.1:3303
      lag: 0.0001838207244873
    name: instance003
    downstream:
      status: follow
      idle: 0.33186100000012
      vclock: {1: 21}
      lag: 0
        ...
```

### Performing manual failover

This section shows how to perform manual failover and change a replica set leader.

#### Switching instances to read-only mode

1. In the `config.yaml` file, change the replica set leader from `instance001` to `null`:

```
replicaset001:
  leader: null
```

2. Reload configurations on all three instances using *config:reload()* and check that instances are in read-only mode. The example below shows how to do this for `instance001`:

```
manual_leader:instance001> require('config'):reload()
---
...
manual_leader:instance001> box.info.ro
---
- true
...
manual_leader:instance001> box.info.ro_reason
---
- config
...
```

3. Make sure that *box.info.vclock* values are the same on all instances:

   - `instance001`:

     ```
     manual_leader:instance001> box.info.vclock
     ---
     - {1: 21}
     ...
     ```

   - `instance002`:

     ```
     manual_leader:instance002> box.info.vclock
     ---
     - {1: 21}
     ...
     ```

   - `instance003`:

     ```
     manual_leader:instance003> box.info.vclock
     ---
     - {1: 21}
     ...
     ```

### Configuring a new leader

1. Change a replica set leader in `config.yaml` to `instance002`:

   ```
   replicaset001:
     leader: instance002
   ```

2. Reload configuration on all instances using *config:reload()*.

3. Make sure that `instance002` is a new master:

   ```
   manual_leader:instance002> box.info.ro
   ---
   - false
   ...
   ```

4. Check replication status using `box.info.replication`.

### Removing instances

This section describes the process of removing an instance from a replica set.

Before removing an instance, make sure it is in read-only mode. If the instance is a master, perform *manual failover*.

### Disconnecting an instance

1. Clear the `iproto` option for `instance003` by setting its value to `{}`:

```
instance003:
  iproto: {}
```

2. Reload configurations on `instance001` and `instance002`:

   - `instance001`:

```
manual_leader:instance001> require('config'):reload()
---
...
```

   - `instance002`:

```
manual_leader:instance002> require('config'):reload()
---
...
```

3. Check that the `upstream` section is missing for `instance003` by executing `box.info.replication[3]`:

```
manual_leader:instance001> box.info.replication[3]
---
- id: 3
  uuid: 9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6
  lsn: 0
  downstream:
    status: follow
    idle: 0.4588760000006
    vclock: {1: 21}
    lag: 0
  name: instance003
...
```

### Stopping an instance

1. Stop `instance003` using the *tt stop* command:

```
$ tt stop manual_leader:instance003
   • The Instance manual_leader:instance003 (PID = 15551) has been terminated.
```

2. Check that `downstream.status` is `stopped` for `instance003`:

```
manual_leader:instance001> box.info.replication[3]
---
- id: 3
  uuid: 9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6
  lsn: 0
  downstream:
    status: stopped
    message: 'unexpected EOF when reading from socket, called on fd 27, aka 127.0.0.
↪1:3301,
      peer of 127.0.0.1:54185: Broken pipe'
    system_message: Broken pipe
  name: instance003
...
```

### Removing an instance from the configuration

1. Remove `instance003` from the `instances.yml` file:

```
instance001:
instance002:
```

2. Remove `instance003` from `config.yaml`:

```
instances:
  instance001:
    iproto:
      listen:
      - uri: '127.0.0.1:3301'
  instance002:
    iproto:
      listen:
      - uri: '127.0.0.1:3302'
```

3. Reload configurations on `instance001` and `instance002`:

   - `instance001`:

   ```
   manual_leader:instance001> require('config'):reload()
   ---
   ...
   ```

   - `instance002`:

   ```
   manual_leader:instance002> require('config'):reload()
   ---
   ...
   ```

### Removing an instance from the „_cluster“ space

To remove an instance from the replica set permanently, it should be removed from the *box.space._cluster* system space:

1. Select all the tuples in the `box.space._cluster` system space:

```
manual_leader:instance002> box.space._cluster:select{}
---
- - [1, '9bb111c2-3ff5-36a7-00f4-2b9a573ea660', 'instance001']
  - [2, '4cfa6e3c-625e-b027-00a7-29b2f2182f23', 'instance002']
  - [3, '9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6', 'instance003']
...
```

2. Delete a tuple corresponding to `instance003`:

```
manual_leader:instance002> box.space._cluster:delete(3)
---
- [3, '9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6', 'instance003']
...
```

3. Execute `box.info.replication` to check the health status:

```
manual_leader:instance002> box.info.replication
---
- 1:
    id: 1
    uuid: 9bb111c2-3ff5-36a7-00f4-2b9a573ea660
    lsn: 21
    upstream:
      status: follow
      idle: 0.73316000000159
      peer: replicator@127.0.0.1:3301
      lag: 0.00016212463378906
    name: instance001
    downstream:
      status: follow
      idle: 0.7269320000014
      vclock: {2: 1, 1: 21}
      lag: 0.00083398818969727
  2:
    id: 2
    uuid: 4cfa6e3c-625e-b027-00a7-29b2f2182f23
    lsn: 1
    name: instance002
...
```

### 3.3.2 Master-replica: automated failover

**Example on GitHub**: auto_leader

This tutorial shows how to configure and work with a replica set with automated failover.

#### Prerequisites

Before starting this tutorial:

1. Install the *tt* utility.
2. Create a tt environment in the current directory by executing the *tt init* command.
3. Inside the `instances.enabled` directory of the created tt environment, create the `auto_leader` directory.
4. Inside `instances.enabled/auto_leader`, create the `instances.yml` and `config.yaml` files:
   - `instances.yml` specifies instances to *run* in the current environment and should look like this:

     ```
     instance001:
     instance002:
     instance003:
     ```

   - The `config.yaml` file is intended to store a *replica set configuration*.

#### Configuring a replica set

This section describes how to configure a replica set in `config.yaml`.

#### Step 1: Configuring a failover mode

First, set the *replication.failover* option to `election`:

```
replication:
  failover: election
```

#### Step 2: Defining a replica set topology

Define a replica set topology inside the *groups* section. The *iproto.listen* option specifies an address used to listen for incoming requests and allows replicas to communicate with each other.

```
groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
```

### Step 3: Creating a user for replication

In the *credentials* section, create the `replicator` user with the `replication` role:

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]
```

### Step 4: Specifying advertise URIs

Set *iproto.advertise.peer* to advertise the current instance to other replica set members:

```
iproto:
  advertise:
    peer:
      login: replicator
```

### Resulting configuration

The resulting replica set configuration should look as follows:

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]

iproto:
  advertise:
    peer:
      login: replicator

replication:
  failover: election

groups:
  group001:
    replicasets:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    replicaset001:
      instances:
        instance001:
          iproto:
            listen:
            - uri: '127.0.0.1:3301'
        instance002:
          iproto:
            listen:
            - uri: '127.0.0.1:3302'
        instance003:
          iproto:
            listen:
            - uri: '127.0.0.1:3303'
```

### Working with a replica set

#### Starting instances

1. After configuring a replica set, execute the *tt start* command from the *tt environment directory*:

```
$ tt start auto_leader
   • Starting an instance [auto_leader:instance001]...
   • Starting an instance [auto_leader:instance002]...
   • Starting an instance [auto_leader:instance003]...
```

2. Check that instances are in the `RUNNING` status using the *tt status* command:

```
$ tt status auto_leader
INSTANCE                    STATUS      PID
auto_leader:instance001     RUNNING     24768
auto_leader:instance002     RUNNING     24769
auto_leader:instance003     RUNNING     24767
```

#### Checking a replica set status

1. Connect to `instance001` using *tt connect*:

```
$ tt connect auto_leader:instance001
   • Connecting to the instance...
   • Connected to auto_leader:instance001
```

2. Check the instance state in regard to *leader election* using *box.info.election*. The output below shows that `instance001` is a follower while `instance002` is a replica set leader.

```
auto_leader:instance001> box.info.election
---
- leader_idle: 0.77491499999815
  leader_name: instance002
  state: follower
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  vote: 0
  term: 2
  leader: 1
...
```

3. Check that `instance001` is in read-only mode using `box.info.ro`:

```
auto_leader:instance001> box.info.ro
---
- true
...
```

4. Execute `box.info.replication` to check a replica set status. Make sure that `upstream.status` and `downstream.status` are `follow` for `instance002` and `instance003`.

```
auto_leader:instance001> box.info.replication
---
- 1:
    id: 1
    uuid: 4cfa6e3c-625e-b027-00a7-29b2f2182f23
    lsn: 9
    upstream:
      status: follow
      idle: 0.8257709999998
      peer: replicator@127.0.0.1:3302
      lag: 0.00012326240539551
    name: instance002
    downstream:
      status: follow
      idle: 0.81174199999805
      vclock: {1: 9}
      lag: 0
  2:
    id: 2
    uuid: 9bb111c2-3ff5-36a7-00f4-2b9a573ea660
    lsn: 0
    name: instance001
  3:
    id: 3
    uuid: 9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6
    lsn: 0
    upstream:
      status: follow
      idle: 0.83125499999733
      peer: replicator@127.0.0.1:3303
      lag: 0.00010204315185547
    name: instance003
    downstream:
      status: follow
      idle: 0.83213399999659
      vclock: {1: 9}
      lag: 0
...
```

To see the diagrams that illustrate how the `upstream` and `downstream` connections look, refer to *Monitoring a replica set*.

### Adding data

To check that replicas (`instance001` and `instance003`) get all updates from the master (`instance002`), follow the steps below:

1. Connect to `instance002` using `tt connect`:

```
$ tt connect auto_leader:instance002
   • Connecting to the instance...
   • Connected to auto_leader:instance002
```

2. Create a space and add data as described in *CRUD operation examples*.
3. Use the `select` operation on `instance001` and `instance003` to make sure data is replicated.
4. Check that the `1` component of *box.info.vclock* values are the same on all instances:
   - `instance001`:

```
auto_leader:instance001> box.info.vclock
---
- {0: 1, 1: 32}
...
```

   - `instance002`:

```
auto_leader:instance002> box.info.vclock
---
- {0: 1, 1: 32}
...
```

   - `instance003`:

```
auto_leader:instance003> box.info.vclock
---
- {0: 1, 1: 32}
...
```

---

**Примечание:** Note that a `vclock` value might include the `0` component that is related to local space operations and might differ for different instances in a replica set.

---

### Testing automated failover

To test how automated failover works if the current master is stopped, follow the steps below:

1. Stop the current master instance (`instance002`) using the `tt stop` command:

```
$ tt stop auto_leader:instance002
   • The Instance auto_leader:instance002 (PID = 24769) has been terminated.
```

2. On `instance001`, check `box.info.election`. In this example, a new replica set leader is `instance001`.

```
auto_leader:instance001> box.info.election
---
- leader_idle: 0
  leader_name: instance001
  state: leader
  vote: 2
  term: 3
  leader: 2
...
```

3. Check replication status using `box.info.replication` for `instance002`:

   - `upstream.status` is `disconnected`.
   - `downstream.status` is `stopped`.

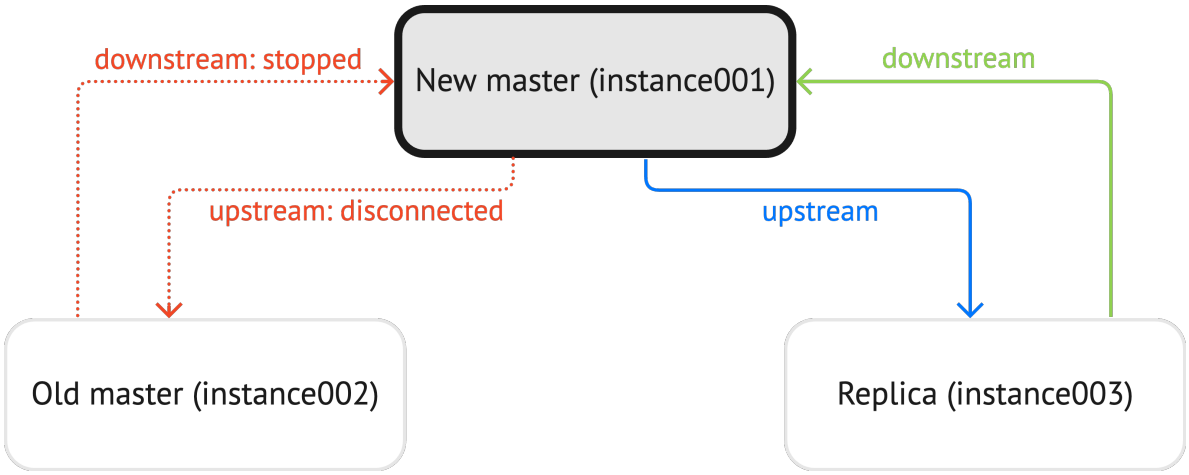
```
auto_leader:instance001> box.info.replication
---
- 1:
    id: 1
    uuid: 4cfa6e3c-625e-b027-00a7-29b2f2182f23
    lsn: 32
    upstream:
      peer: replicator@127.0.0.1:3302
      lag: 0.00032305717468262
      status: disconnected
      idle: 48.352504000002
      message: 'connect, called on fd 20, aka 127.0.0.1:62575: Connection refused'
      system_message: Connection refused
    name: instance002
    downstream:
      status: stopped
      message: 'unexpected EOF when reading from socket, called on fd 32, aka 127.0.
↪0.1:3301,
        peer of 127.0.0.1:62204: Broken pipe'
      system_message: Broken pipe
  2:
    id: 2
    uuid: 9bb111c2-3ff5-36a7-00f4-2b9a573ea660
    lsn: 1
    name: instance001
  3:
    id: 3
    uuid: 9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    lsn: 0
    upstream:
      status: follow
      idle: 0.18620999999985
      peer: replicator@127.0.0.1:3303
      lag: 0.00012516975402832
    name: instance003
    downstream:
      status: follow
      idle: 0.19718099999955
      vclock: {2: 1, 1: 32}
      lag: 0.00051403045654297
...
```

The diagram below illustrates how the `upstream` and `downstream` connections look like:




4. Start `instance002` back using `tt start`:

```
$ tt start auto_leader:instance002
   • Starting an instance [auto_leader:instance002]...
```

### Choosing a leader manually

1. Make sure that *box.info.vclock* values (except the `0` components) are the same on all instances:

   - `instance001`:

     ```
     auto_leader:instance001> box.info.vclock
     ---
     - {0: 2, 1: 32, 2: 1}
     ...
     ```

   - `instance002`:

     ```
     auto_leader:instance002> box.info.vclock
     ---
     ```

(continues on next page)

(продолжение с предыдущей страницы)

```
- {0: 2, 1: 32, 2: 1}
...
```

- `instance003`:

```
auto_leader:instance003> box.info.vclock
---
- {0: 3, 1: 32, 2: 1}
...
```

2. On `instance002`, run *box.ctl.promote()* to choose it as a new replica set leader:

```
auto_leader:instance002> box.ctl.promote()
---
...
```

3. Check `box.info.election` to make sure `instance002` is a leader now:

```
auto_leader:instance002> box.info.election
---
- leader_idle: 0
  leader_name: instance002
  state: leader
  vote: 1
  term: 4
  leader: 1
...
```

#### Adding and removing instances

The process of adding instances to a replica set and removing them is similar for all failover modes. Learn how to do this from the *Master-replica: manual failover* tutorial:

- *Adding instances*
- *Removing instances*

Before removing an instance from a replica set with *replication.failover* set to `election`, make sure this instance is in read-only mode. If the instance is a master, choose a *new leader manually*.

### 3.3.3 Master-master

**Example on GitHub**: master_master

This tutorial shows how to configure and work with a master-master replica set.

### Prerequisites

Before starting this tutorial:

1. Install the *tt* utility.
2. Create a tt environment in the current directory by executing the *tt init* command.
3. Inside the `instances.enabled` directory of the created tt environment, create the `master_master` directory.
4. Inside `instances.enabled/master_master`, create the `instances.yml` and `config.yaml` files:
   - `instances.yml` specifies instances to *run* in the current environment and should look like this:

     ```
     instance001:
     instance002:
     ```

   - The `config.yaml` file is intended to store a *replica set configuration*.

### Configuring a replica set

This section describes how to configure a replica set in `config.yaml`.

#### Step 1: Configuring a failover mode

First, set the *replication.failover* option to `off`:

```
replication:
  failover: off
```

#### Step 2: Defining a replica set topology

Define a replica set topology inside the *groups* section:

- The `database.mode` option should be set to `rw` to make instances work in read-write mode.
- The *iproto.listen* option specifies an address used to listen for incoming requests and allows replicas to communicate with each other.

```
groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            database:
              mode: rw
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            database:
              mode: rw
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
```

### Step 3: Creating a user for replication

In the *credentials* section, create the `replicator` user with the `replication` role:

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]
```

### Step 4: Specifying advertise URIs

Set *iproto.advertise.peer* to advertise the current instance to other replica set members:

```
iproto:
  advertise:
    peer:
      login: replicator
```

### Resulting configuration

The resulting replica set configuration should look as follows:

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]

iproto:
  advertise:
    peer:
      login: replicator

replication:
  failover: off

groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            database:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            mode: rw
          iproto:
            listen:
            - uri: '127.0.0.1:3301'
        instance002:
          database:
            mode: rw
          iproto:
            listen:
            - uri: '127.0.0.1:3302'
```

### Working with a replica set

#### Starting instances

1. After configuring a replica set, execute the *tt start* command from the *tt environment directory*:

```
$ tt start master_master
   • Starting an instance [master_master:instance001]...
   • Starting an instance [master_master:instance002]...
```

2. Check that instances are in the `RUNNING` status using the *tt status* command:

```
$ tt status master_master
INSTANCE                        STATUS      PID
master_master:instance001       RUNNING     30818
master_master:instance002       RUNNING     30819
```

#### Checking a replica set status

1. Connect to both instances using *tt connect*. Below is the example for `instance001`:

```
$ tt connect master_master:instance001
   • Connecting to the instance...
   • Connected to master_master:instance001

master_master:instance001>
```

2. Check that both instances are writable using `box.info.ro`:

   - `instance001`:

```
master_master:instance001> box.info.ro
---
- false
...
```

   - `instance002`:

```
master_master:instance002> box.info.ro
---
- false
...
```

3. Execute `box.info.replication` to check a replica set status. For `instance002`, `upstream.status` and `downstream.status` should be `follow`.

```
master_master:instance001> box.info.replication
---
- 1:
    id: 1
    uuid: c3bfd89f-5a1c-4556-aa9f-461377713a2a
    lsn: 7
    name: instance001
  2:
    id: 2
    uuid: dccf7485-8bff-47f6-bfc4-b311701e36ef
    lsn: 0
    upstream:
      status: follow
      idle: 0.93246499999987
      peer: replicator@127.0.0.1:3302
      lag: 0.00016188621520996
    name: instance002
    downstream:
      status: follow
      idle: 0.8988360000003
      vclock: {1: 7}
      lag: 0
...
```

To see the diagrams that illustrate how the `upstream` and `downstream` connections look, refer to *Monitoring a replica set*.

---

**Примечание:** Note that a `vclock` value might include the `0` component that is related to local space operations and might differ for different instances in a replica set.

---

### Adding data

To check that both instances get updates from each other, follow the steps below:

1. On `instance001`, create a space, format it, and create a primary index:

```
box.schema.space.create('bands')
box.space.bands:format({
    { name = 'id', type = 'unsigned' },
    { name = 'band_name', type = 'string' },
    { name = 'year', type = 'unsigned' }
})
box.space.bands:create_index('primary', { parts = { 'id' } })
```

Then, add sample data to this space:

```
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
```

2. On `instance002`, use the `select` operation to make sure data is replicated:

```
master_master:instance002> box.space.bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
...
```

3. Add more data to the created space on `instance002`:

```
box.space.bands:insert { 3, 'Ace of Base', 1987 }
box.space.bands:insert { 4, 'The Beatles', 1960 }
```

4. Get back to `instance001` and use `select` to make sure new records are replicated:

```
master_master:instance001> box.space.bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
...
```

5. Check that *box.info.vclock* values are the same on both instances:

   - `instance001`:

   ```
   master_master:instance001> box.info.vclock
   ---
   - {2: 2, 1: 12}
   ...
   ```

   - `instance002`:

   ```
   master_master:instance002> box.info.vclock
   ---
   - {2: 2, 1: 12}
   ...
   ```

### Resolving replication conflicts

**Примечание:** To learn how to fix and prevent replication conflicts using trigger functions, see *Resolving replication conflicts*.

### Inserting conflicting records

To insert conflicting records to `instance001` and `instance002`, follow the steps below:

1. Stop `instance001` using the `tt stop` command:

```
$ tt stop master_master:instance001
```

2. On `instance002`, insert a new record:

```
box.space.bands:insert { 5, 'incorrect data', 0 }
```

3. Stop `instance002` using `tt stop`:

```
$ tt stop master_master:instance002
```

4. Start `instance001` back:

```
$ tt start master_master:instance001
```

5. Connect to `instance001` and insert a record that should conflict with a record already inserted on `instance002`:

```
box.space.bands:insert { 5, 'Pink Floyd', 1965 }
```

6. Start `instance002` back:

```
$ tt start master_master:instance002
```

Then, check `box.info.replication` on `instance001`. `upstream.status` should be `stopped` because of the `Duplicate key exists` error:

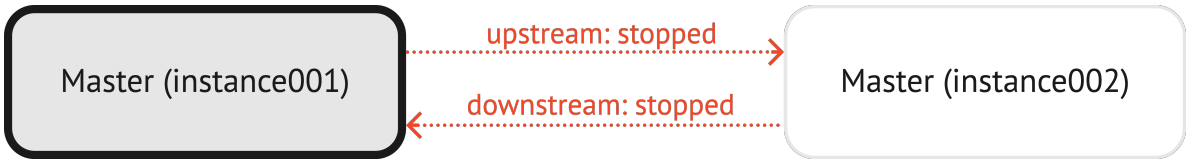
```
master_master:instance001> box.info.replication
---
- 1:
    id: 1
    uuid: c3bfd89f-5a1c-4556-aa9f-461377713a2a
    lsn: 13
    name: instance001
  2:
    id: 2
    uuid: dccf7485-8bff-47f6-bfc4-b311701e36ef
    lsn: 2
    upstream:
      peer: replicator@127.0.0.1:3302
      lag: 115.99977827072
      status: stopped
      idle: 2.0342070000006
      message: Duplicate key exists in unique index "primary" in space "bands" with
        old tuple - [5, "Pink Floyd", 1965] and new tuple - [5, "incorrect data",
        0]
    name: instance002
    downstream:
      status: stopped
      message: 'unexpected EOF when reading from socket, called on fd 24, aka 127.0.
↪0.1:3301,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        peer of 127.0.0.1:58478: Broken pipe'
      system_message: Broken pipe
...
```

The diagram below illustrates how the `upstream` and `downstream` connections look like:




### Reseeding a replica

To resolve a replication conflict, `instance002` should get the correct data from `instance001` first. To achieve this, `instance002` should be rebootstrapped:

1. Select all the tuples in the *box.space._cluster* system space to get a UUID of `instance002`:

```
master_master:instance001> box.space._cluster:select()
---
- - [1, 'c3bfd89f-5a1c-4556-aa9f-461377713a2a', 'instance001']
  - [2, 'dccf7485-8bff-47f6-bfc4-b311701e36ef', 'instance002']
...
```

2. In the `config.yaml` file, change the following `instance002` settings:
   - Set `database.mode` to `ro`.
   - Set `database.instance_uuid` to a UUID value obtained in the previous step.

```
instance002:
  database:
    mode: ro
    instance_uuid: 'dccf7485-8bff-47f6-bfc4-b311701e36ef'
```

3. Reload configurations on both instances using the *config:reload()* function:
   - `instance001`:

```
master_master:instance001> require('config'):reload()
---
...
```

   - `instance002`:

```
master_master:instance002> require('config'):reload()
---
...
```

4. Delete write-ahead logs and snapshots stored in the `var/lib/instance002` directory.

---

**Примечание:** `var/lib` is the default directory used by tt to store write-ahead logs and snapshots. Learn more from *Configuration*.

---

5. Restart `instance002` using the *tt restart* command:

```
$ tt restart master_master:instance002
```

6. Connect to `instance002` and make sure it received the correct data from `instance001`:

```
master_master:instance002> box.space.bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
  - [5, 'Pink Floyd', 1965]
...
```

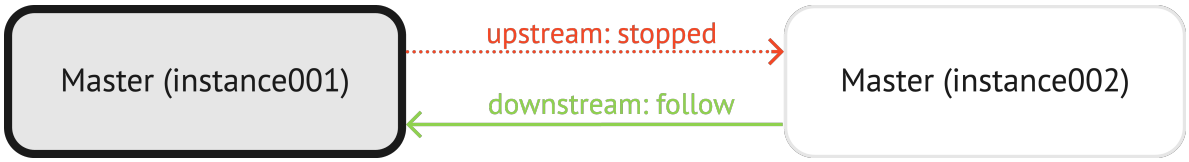
### Restarting replication

After *reseeding a replica*, you need to resolve a replication conflict that keeps replication stopped:

1. Execute `box.info.replication` on `instance001`. `upstream.status` is still stopped:

```
master_master:instance001> box.info.replication
---
- 1:
    id: 1
    uuid: c3bfd89f-5a1c-4556-aa9f-461377713a2a
    lsn: 13
    name: instance001
  2:
    id: 2
    uuid: dccf7485-8bff-47f6-bfc4-b311701e36ef
    lsn: 2
    upstream:
      peer: replicator@127.0.0.1:3302
      lag: 115.99977827072
      status: stopped
      idle: 1013.688243
      message: Duplicate key exists in unique index "primary" in space "bands" with
        old tuple - [5, "Pink Floyd", 1965] and new tuple - [5, "incorrect data",
        0]
    name: instance002
    downstream:
      status: follow
      idle: 0.69694700000036
      vclock: {2: 2, 1: 13}
      lag: 0
...
```

The diagram below illustrates how the `upstream` and `downstream` connections look like:




2. In the `config.yaml` file, clear the `iproto` option for `instance001` by setting its value to `{}` to disconnect this instance from `instance002`. Set `database.mode` to `ro`:

```
instance001:
  database:
    mode: ro
  iproto: {}
```

3. Reload configuration on `instance001` only:

```
master_master:instance001> require('config'):reload()
---
...
```

4. Change `database.mode` values back to `rw` for both instances and restore `iproto.listen` for `instance001`. The `database.instance_uuid` option can be removed for `instance002`:

```
instance001:
  database:
    mode: rw
  iproto:
    listen:
    - uri: '127.0.0.1:3301'
instance002:
  database:
    mode: rw
  iproto:
    listen:
    - uri: '127.0.0.1:3302'
```

5. Reload configurations on both instances one more time:

   - `instance001`:

     ```
     master_master:instance001> require('config'):reload()
     ---
     ...
     ```

   - `instance002`:

     ```
     master_master:instance002> require('config'):reload()
     ---
     ...
     ```

6. Check `box.info.replication`. `upstream.status` should be `follow` now.

```
master_master:instance001> box.info.replication
---
- 1:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    id: 1
    uuid: c3bfd89f-5a1c-4556-aa9f-461377713a2a
    lsn: 13
    name: instance001
  2:
    id: 2
    uuid: dccf7485-8bff-47f6-bfc4-b311701e36ef
    lsn: 2
    upstream:
      status: follow
      idle: 0.86873800000012
      peer: replicator@127.0.0.1:3302
      lag: 0.0001060962677002
    name: instance002
    downstream:
      status: follow
      idle: 0.058662999999797
      vclock: {2: 2, 1: 13}
      lag: 0
...
```

### Adding and removing instances

The process of adding instances to a replica set and removing them is similar for all failover modes. Learn how to do this from the *Master-replica: manual failover* tutorial:

- *Adding instances*
- *Removing instances*

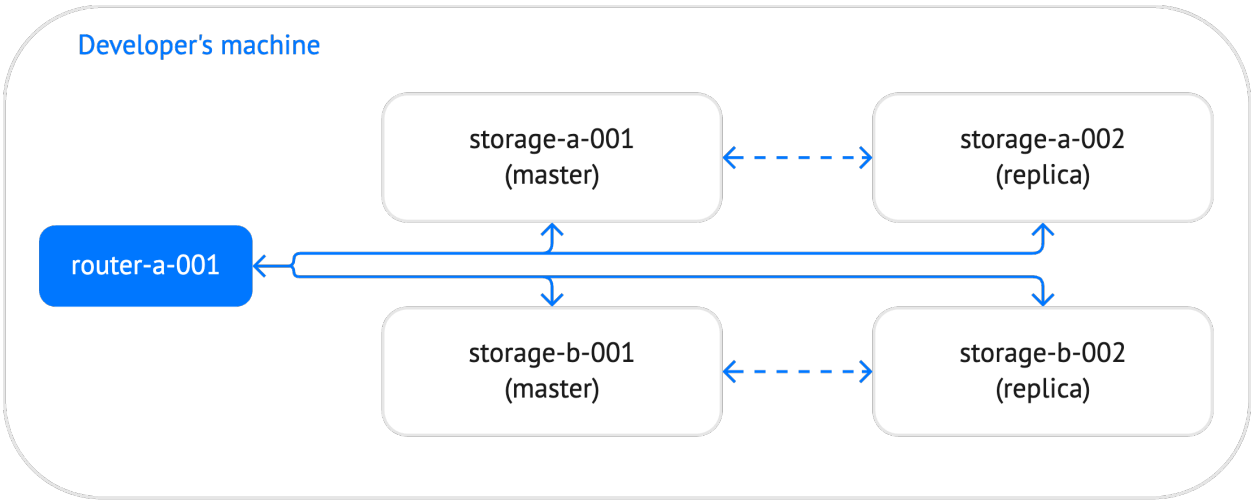
Before removing an instance from a replica set with *replication.failover* set to `off`, make sure this instance is in read-only mode.

## 3.4 Creating a sharded cluster

**Example on GitHub**: sharded_cluster

In this tutorial, you get a sharded cluster up and running on your local machine and learn how to manage the cluster using the tt utility. To enable sharding in the cluster, the *vshard* module is used.

The cluster created in this tutorial includes 5 instances: one router and 4 storages, which constitute two replica sets.




### 3.4.1 Prerequisites

Before starting this tutorial:

- *Install the tt* utility.
- Install tarantool.

---

**Примечание:** The tt utility provides the ability to install Tarantool software using the *tt install* command.

---

### 3.4.2 Creating a cluster application

The *tt create* command can be used to create an application from a predefined or custom template. For example, the built-in `vshard_cluster` template enables you to create a ready-to-run sharded cluster application.

In this tutorial, the application layout is prepared manually:

1. Create a tt environment in the current directory by executing the *tt init* command.
2. Inside the empty `instances.enabled` directory of the created tt environment, create the `sharded_cluster` directory.
3. Inside `instances.enabled/sharded_cluster`, create the following files:
   - `instances.yml` specifies instances to run in the current environment.
   - `config.yaml` specifies the cluster's *configuration*.
   - `storage.lua` contains code specific for *storages*.
   - `router.lua` contains code specific for a *router*.
   - `sharded_cluster-scm-1.rockspec` specifies external dependencies required by the application.

   The next *Developing the application* section shows how to configure the cluster and write code for routing read and write requests to different storages.

### 3.4.3 Developing the application

#### Configuring instances to run

Open the `instances.yml` file and add the following content:

```
storage-a-001:
storage-a-002:
storage-b-001:
storage-b-002:
router-a-001:
```

This file specifies instances to run in the current environment.

#### Configuring the cluster

This section describes how to configure the cluster in the `config.yaml` file.

#### Step 1: Configuring credentials

Add the *credentials* configuration section:

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]
    storage:
      password: 'secret'
      roles: [sharding]
```

In this section, two users with the specified passwords are created:

- The `replicator` user with the `replication` role.
- The `storage` user with the `sharding` role.

These users are intended to maintain replication and sharding in the cluster.

---

**Важно:** It is not recommended to store passwords as plain text in a YAML configuration. Learn how to load passwords from safe storage such as external files or environment variables from *Loading secrets from safe storage*.

---

### Step 2: Specifying advertise URIs

Add the *iproto.advertise* section:

```
iproto:
  advertise:
    peer:
      login: replicator
    sharding:
      login: storage
```

In this section, the following options are configured:

- `iproto.advertise.peer` specifies how to advertise the current instance to other cluster members. In particular, this option informs other replica set members that the `replicator` user should be used to connect to the current instance.
- `iproto.advertise.sharding` specifies how to advertise the current instance to a router and rebalancer.

### Step 3: Configuring bucket count

Specify the total number of *buckets* in a sharded cluster using the *sharding.bucket_count* option:

```
sharding:
  bucket_count: 1000
```

### Step 4: Defining the cluster topology

Define the cluster's topology inside the *groups* section. The cluster includes two groups:

- `storages` includes two replica sets. Each replica set contains two instances.
- `routers` includes one router instance.

Here is a schematic view of the cluster's topology:

```
groups:
  storages:
    replicasets:
      storage-a:
        # ...
      storage-b:
        # ...
  routers:
    replicasets:
      router-a:
        # ...
```

1. To configure storages, add the following code inside the `groups` section:

   ```
   storages:
     app:
       module: storage
     sharding:
   ```

(continues on next page)

(продолжение с предыдущей страницы)

```
    roles: [storage]
  replication:
    failover: manual
  replicasets:
    storage-a:
      leader: storage-a-001
      instances:
        storage-a-001:
          iproto:
            listen:
            - uri: '127.0.0.1:3302'
        storage-a-002:
          iproto:
            listen:
            - uri: '127.0.0.1:3303'
    storage-b:
      leader: storage-b-001
      instances:
        storage-b-001:
          iproto:
            listen:
            - uri: '127.0.0.1:3304'
        storage-b-002:
          iproto:
            listen:
            - uri: '127.0.0.1:3305'
```

The main group-level options here are:

- `app`: The `app.module` option specifies that code specific to storages should be loaded from the `storage` module. This is explained below in the *Adding storage code* section.
- `sharding`: The *sharding.roles* option specifies that all instances inside this group act as storages. A rebalancer is selected automatically from two master instances.
- `replication`: The *replication.failover* option specifies that a leader in each replica set should be specified manually.
- `replicasets`: This section configures two replica sets that constitute cluster storages.

2. To configure a router, add the following code inside the `groups` section:

```
routers:
  app:
    module: router
  sharding:
    roles: [router]
  replicasets:
    router-a:
      instances:
        router-a-001:
          iproto:
            listen:
            - uri: '127.0.0.1:3301'
```

The main group-level options here are:

- `app`: The `app.module` option specifies that code specific to a router should be loaded from the `router` module. This is explained below in the *Adding router code* section.
- `sharding`: The *sharding.roles* option specifies that an instance inside this group acts as a router.
- `replicasets`: This section configures one replica set with one router instance.

### Resulting configuration

The resulting `config.yaml` file should look as follows:

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]
    storage:
      password: 'secret'
      roles: [sharding]

iproto:
  advertise:
    peer:
      login: replicator
    sharding:
      login: storage

sharding:
  bucket_count: 1000

groups:
  storages:
    app:
      module: storage
    sharding:
      roles: [storage]
    replication:
      failover: manual
    replicasets:
      storage-a:
        leader: storage-a-001
        instances:
          storage-a-001:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          storage-a-002:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
      storage-b:
        leader: storage-b-001
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      instances:
        storage-b-001:
          iproto:
            listen:
            - uri: '127.0.0.1:3304'
        storage-b-002:
          iproto:
            listen:
            - uri: '127.0.0.1:3305'
routers:
  app:
    module: router
  sharding:
    roles: [router]
  replicasets:
    router-a:
      instances:
        router-a-001:
          iproto:
            listen:
            - uri: '127.0.0.1:3301'
```

### Adding storage code

1. Open the `storage.lua` file and create a space using the *box.schema.space.create()* function:

   ```
   box.schema.create_space('bands', {
       format = {
           { name = 'id', type = 'unsigned' },
           { name = 'bucket_id', type = 'unsigned' },
           { name = 'band_name', type = 'string' },
           { name = 'year', type = 'unsigned' }
       },
       if_not_exists = true
   })
   ```

   Note that the created `bands` spaces includes the `bucket_id` field. This field represents a sharding key used to partition a dataset across different storage instances.

2. Create two indexes based on the `id` and `bucket_id` fields:

   ```
   box.space.bands:create_index('id', { parts = { 'id' }, if_not_exists = true })
   box.space.bands:create_index('bucket_id', { parts = { 'bucket_id' }, unique = false,
   ↪ if_not_exists = true })
   ```

3. Define the `insert_band` function that inserts a tuple into the created space:

   ```
   function insert_band(id, bucket_id, band_name, year)
       box.space.bands:insert({ id, bucket_id, band_name, year })
   end
   ```

4. Define the `get_band` function that returns data without the `bucket_id` value:

```
function get_band(id)
    local tuple = box.space.bands:get(id)
    if tuple == nil then
        return nil
    end
    return { tuple.id, tuple.band_name, tuple.year }
end
```

The resulting `storage.lua` file should look as follows:

```
box.schema.create_space('bands', {
    format = {
        { name = 'id', type = 'unsigned' },
        { name = 'bucket_id', type = 'unsigned' },
        { name = 'band_name', type = 'string' },
        { name = 'year', type = 'unsigned' }
    },
    if_not_exists = true
})
box.space.bands:create_index('id', { parts = { 'id' }, if_not_exists = true })
box.space.bands:create_index('bucket_id', { parts = { 'bucket_id' }, unique = false, if_
↪not_exists = true })

function insert_band(id, bucket_id, band_name, year)
    box.space.bands:insert({ id, bucket_id, band_name, year })
end

function get_band(id)
    local tuple = box.space.bands:get(id)
    if tuple == nil then
        return nil
    end
    return { tuple.id, tuple.band_name, tuple.year }
end
```

### Adding router code

1. Open the `router.lua` file and load the `vshard` module as follows:

   ```
   local vshard = require('vshard')
   ```

2. Define the `put` function that specifies how the router selects the storage to write data:

   ```
   function put(id, band_name, year)
       local bucket_id = vshard.router.bucket_id_mpcrc32({ id })
       vshard.router.callrw(bucket_id, 'insert_band', { id, bucket_id, band_name, year␣
   ↪})
   end
   ```

   The following `vshard` router functions are used:

   - *vshard.router.bucket_id_mpcrc32()*: Calculates a bucket ID value using a hash function.
   - *vshard.router.callrw()*: Inserts a tuple to a storage identified the generated bucket ID.

3. Create the get function for getting data:

```
function get(id)
    local bucket_id = vshard.router.bucket_id_mpcrc32({ id })
    return vshard.router.callro(bucket_id, 'get_band', { id })
end
```

   Inside this function, *vshard.router.callro()* is called to get data from a storage identified the generated bucket ID.

4. Finally, create the `insert_data()` function that inserts sample data into the created space:

```
function insert_data()
    put(1, 'Roxette', 1986)
    put(2, 'Scorpions', 1965)
    put(3, 'Ace of Base', 1987)
    put(4, 'The Beatles', 1960)
    put(5, 'Pink Floyd', 1965)
    put(6, 'The Rolling Stones', 1962)
    put(7, 'The Doors', 1965)
    put(8, 'Nirvana', 1987)
    put(9, 'Led Zeppelin', 1968)
    put(10, 'Queen', 1970)
end
```

The resulting `router.lua` file should look as follows:

```
local vshard = require('vshard')

function put(id, band_name, year)
    local bucket_id = vshard.router.bucket_id_mpcrc32({ id })
    vshard.router.callrw(bucket_id, 'insert_band', { id, bucket_id, band_name, year })
end

function get(id)
    local bucket_id = vshard.router.bucket_id_mpcrc32({ id })
    return vshard.router.callro(bucket_id, 'get_band', { id })
end

function insert_data()
    put(1, 'Roxette', 1986)
    put(2, 'Scorpions', 1965)
    put(3, 'Ace of Base', 1987)
    put(4, 'The Beatles', 1960)
    put(5, 'Pink Floyd', 1965)
    put(6, 'The Rolling Stones', 1962)
    put(7, 'The Doors', 1965)
    put(8, 'Nirvana', 1987)
    put(9, 'Led Zeppelin', 1968)
    put(10, 'Queen', 1970)
end
```

#### Configuring build settings

Open the `sharded_cluster-scm-1.rockspec` file and add the following content:

```
package = 'sharded_cluster'
version = 'scm-1'
source  = {
    url = '/dev/null',
}

dependencies = {
    'vshard == 0.1.26'
}
build = {
    type = 'none';
}
```

The `dependencies` section includes the specified version of the `vshard` module. To install dependencies, you need to *build the application*.

### 3.4.4 Building the application

In the terminal, open the *tt environment directory*. Then, execute the `tt build` command:

```
$ tt build sharded_cluster
   • Running rocks make
No existing manifest. Attempting to rebuild...
   • Application was successfully built
```

This installs the `vshard` dependency defined in the **.rockspec* file to the `.rocks` directory.

### 3.4.5 Working with the cluster

#### Starting instances

To start all instances in the cluster, execute the `tt start` command:

```
$ tt start sharded_cluster
   • Starting an instance [sharded_cluster:storage-a-001]...
   • Starting an instance [sharded_cluster:storage-a-002]...
   • Starting an instance [sharded_cluster:storage-b-001]...
   • Starting an instance [sharded_cluster:storage-b-002]...
   • Starting an instance [sharded_cluster:router-a-001]...
```

### Bootstrapping a cluster

After starting instances, you need to bootstrap the cluster as follows:

1. Connect to the router instance using `tt connect`:

```
$ tt connect sharded_cluster:router-a-001
   • Connecting to the instance...
   • Connected to sharded_cluster:router-a-001
```

2. Call *vshard.router.bootstrap()* to perform the initial cluster bootstrap:

```
sharded_cluster:router-a-001> vshard.router.bootstrap()
---
- true
...
```

### Checking status

To check the cluster's status, execute *vshard.router.info()* on the router:

```
sharded_cluster:router-a-001> vshard.router.info()
---
- replicasets:
    storage-b:
      replica:
        network_timeout: 0.5
        status: available
        uri: storage@127.0.0.1:3305
        name: storage-b-002
      bucket:
        available_rw: 500
      master:
        network_timeout: 0.5
        status: available
        uri: storage@127.0.0.1:3304
        name: storage-b-001
      name: storage-b
    storage-a:
      replica:
        network_timeout: 0.5
        status: available
        uri: storage@127.0.0.1:3303
        name: storage-a-002
      bucket:
        available_rw: 500
      master:
        network_timeout: 0.5
        status: available
        uri: storage@127.0.0.1:3302
        name: storage-a-001
      name: storage-a
  bucket:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    unreachable: 0
    available_ro: 0
    unknown: 0
    available_rw: 1000
  status: 0
  alerts: []
...
```

The output includes the following sections:

- `replicasets`: contains information about storages and their availability.
- `bucket`: displays the total number of read-write and read-only buckets that are currently available for this router.
- `status`: the number from 0 to 3 that indicates whether there are any issues with the cluster. 0 means that there are no issues.
- `alerts`: might describe the exact issues related to bootstrapping a cluster, for example, connection issues, failover events, or unidentified buckets.

### Writing and selecting data

1. To insert sample data, call the *insert_data()* function on the router:

   ```
   sharded_cluster:router-a-001> insert_data()
   ---
   ...
   ```

   Calling this function *distributes data* evenly across the cluster's nodes.

2. To get a tuple by the specified ID, call the `get()` function:

   ```
   sharded_cluster:router-a-001> get(4)
   ---
   - [4, 'The Beatles', 1960]
   ...
   ```

3. To insert a new tuple, call the `put()` function:

   ```
   sharded_cluster:router-a-001> put(11, 'The Who', 1962)
   ---
   ...
   ```

### Checking data distribution

To check how data is distributed across the cluster's nodes, follow the steps below:

1. Connect to any storage in the `storage-a` replica set:

   ```
   $ tt connect sharded_cluster:storage-a-001
      • Connecting to the instance...
      • Connected to sharded_cluster:storage-a-001
   ```

   Then, select all tuples in the `bands` space:

```
sharded_cluster:storage-a-001> box.space.bands:select()
---
- - [3, 11, 'Ace of Base', 1987]
  - [4, 42, 'The Beatles', 1960]
  - [6, 55, 'The Rolling Stones', 1962]
  - [9, 299, 'Led Zeppelin', 1968]
  - [10, 167, 'Queen', 1970]
  - [11, 70, 'The Who', 1962]
...
```

2. Connect to any storage in the `storage-b` replica set:

```
$ tt connect sharded_cluster:storage-b-001
   • Connecting to the instance...
   • Connected to sharded_cluster:storage-b-001
```

Select all tuples in the `bands` space to make sure it contains another subset of data:

```
sharded_cluster:storage-b-001> box.space.bands:select()
---
- - [1, 614, 'Roxette', 1986]
  - [2, 986, 'Scorpions', 1965]
  - [5, 755, 'Pink Floyd', 1965]
  - [7, 998, 'The Doors', 1965]
  - [8, 762, 'Nirvana', 1987]
...
```

## 3.5 Getting started with Tarantool Cluster Manager

**Enterprise Edition**

This tutorial uses Tarantool Enterprise Edition.

**Example on GitHub**: tcm_get_started

In this tutorial, you get *Tarantool Cluster Manager* up and running on your local system, deploy a local Tarantool EE cluster, and learn to manage the cluster from the TCM web UI.

To complete this tutorial, you need:

- A Linux machine with glibc 2.17 or later.
- A web browser: Chromium-based (Chromium version 108 or later), Mozilla Firefox 101 or later, or another up-to-date browser.
- The Tarantool Enterprise Edition SDK 3.0 or later in the `tar.gz` archive. See *Настройка* for information about getting the archive.

For more detailed information about using TCM, refer to *Tarantool Cluster Manager*.

### 3.5.1 Setting up Tarantool EE

1. Extract the Tarantool EE SDK archive:

```
$ tar -xvzf tarantool-enterprise-sdk-gc64-<VERSION>-<HASH>-r<REVISION>.linux.x86_64.
↪tar.gz
```

   This creates the `tarantool-enterprise` directory beside the archive. The directory contains three executables for key Tarantool EE components:

   - `tarantool` – Tarantool Enterprise Edition.
   - `tt` – the *tt* command-line utility.
   - `tcm` – *Tarantool Cluster Manager*.

2. Add the Tarantool EE components to the executable path by executing the `env.sh` script included in the distribution:

```
$ source tarantool-enterprise/env.sh
```

3. To check that the Tarantool EE executables `tarantool`, `tt`, and `tcm` are available in the system, print their versions:

```
$ tarantool --version
Tarantool Enterprise 3.0.0-0-gf58f7d82a-r23-gc64
Target: Linux-x86_64-RelWithDebInfo
Build options: cmake . -DCMAKE_INSTALL_PREFIX=/home/centos/release/sdk/tarantool/
↪static-build/tarantool-prefix -DENABLE_BACKTRACE=TRUE
Compiler: GNU-9.3.1
C_FLAGS: -fexceptions -funwind-tables -fasynchronous-unwind-tables -static-
↪libstdc++ -fno-common -msse2  -fmacro-prefix-map=/home/centos/release/sdk/
↪tarantool=. -std=c11 -Wall -Wextra -Wno-gnu-alignof-expression -fno-gnu89-inline -
↪Wno-cast-function-type -O2 -g -DNDEBUG -ggdb -O2
CXX_FLAGS: -fexceptions -funwind-tables -fasynchronous-unwind-tables -static-
↪libstdc++ -fno-common -msse2  -fmacro-prefix-map=/home/centos/release/sdk/
↪tarantool=. -std=c++11 -Wall -Wextra -Wno-invalid-offsetof -Wno-gnu-alignof-
↪expression -Wno-cast-function-type -O2 -g -DNDEBUG -ggdb -O2
$ tt version
Tarantool CLI EE 2.1.0, linux/amd64. commit: d80c2e3
$ tcm version
1.0.0-0-gd38b12c2
```

### 3.5.2 Starting TCM

Tarantool Cluster Manager is ready to run out of the box. To start TCM run the following command:

```
$ tcm --storage.etcd.embed.enabled
```

**Важно:** The TCM bootstrap log in the terminal includes a message with the credentials to use for the first login. Make sure to save them somewhere.

```
Jan 24 05:51:28.443 WRN Generated super admin credentials login=admin␣
↪password=qF3A5rjGurjAwmlYccJ7JrL5XqjbIHY6
```

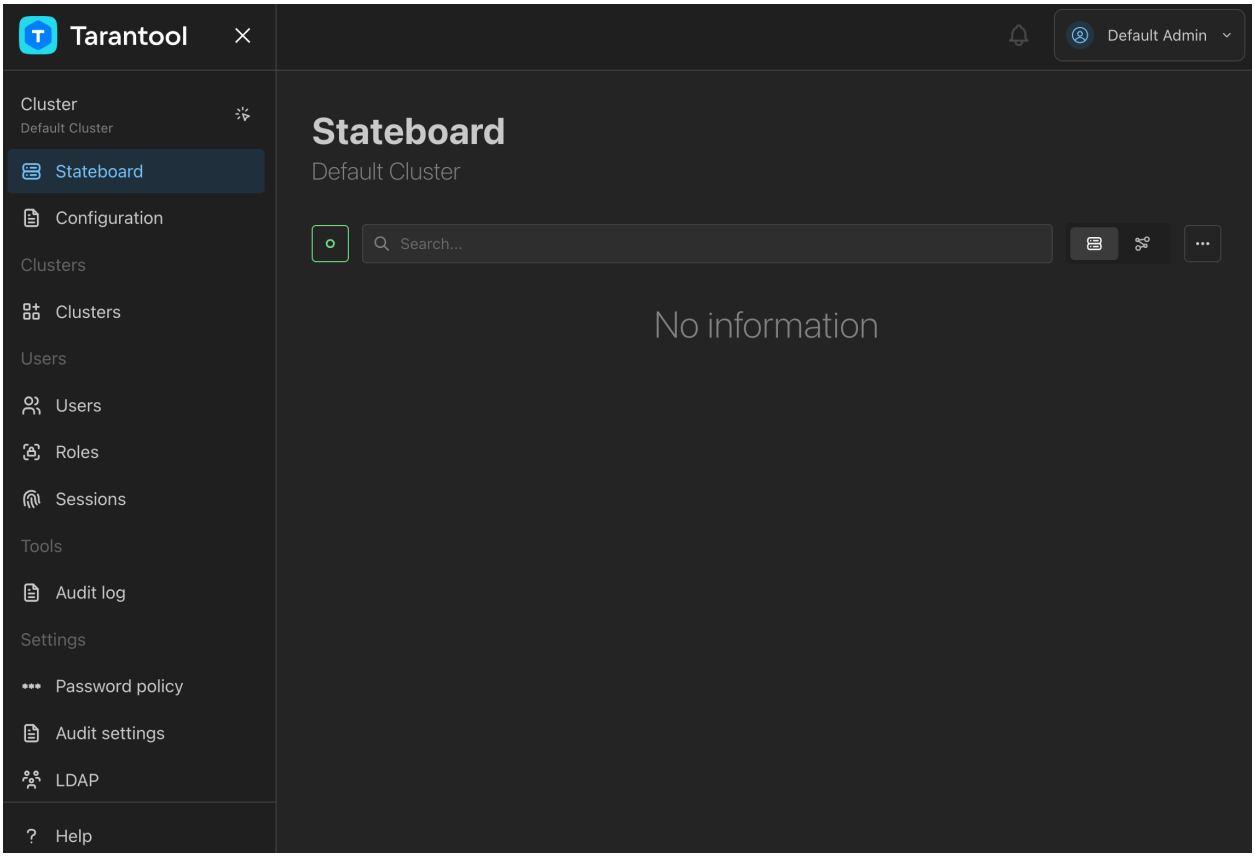
The *–storage.etcd.embed.enabled* option makes TCM start its own instance of etcd on bootstrap. This `etcd` instance is used for storing the *TCM configuration*.

**Примечание:** During the development, it is also convenient to use the TCM-embedded etcd as a configuration storage for Tarantool EE clusters connected to TCM. Learn more in *Centralized configuration storages*.

### 3.5.3 Logging into TCM

1. Open a web browser and go to `http://127.0.0.1:8080/`.
2. Enter the username and the password you got from the TCM bootstrap log in the previous step.
3. Click **Log in**.

After a successful login, you see the TCM web UI:



### 3.5.4 Setting up a Tarantool EE cluster

To prepare a Tarantool EE cluster, complete the following steps:

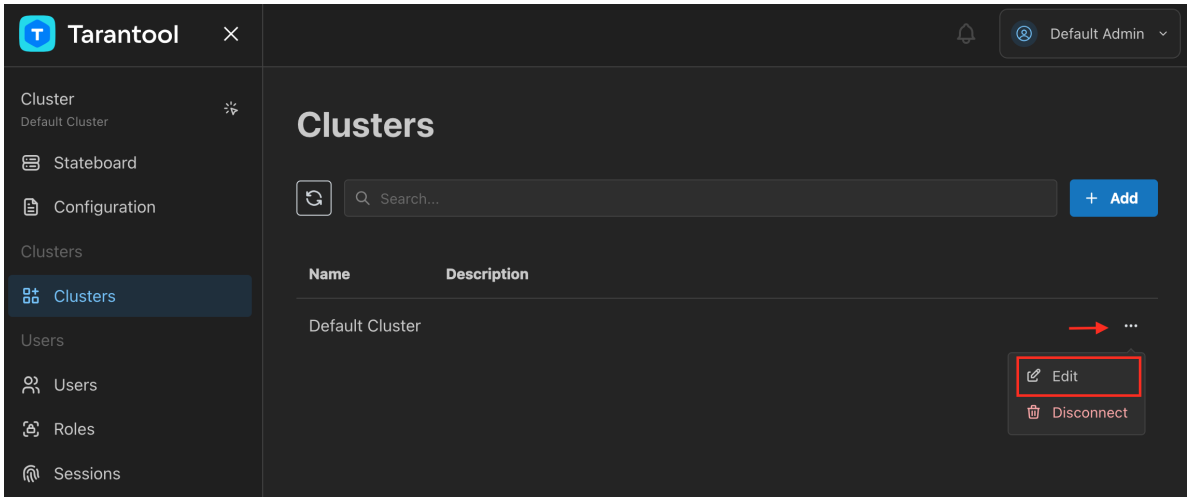
1. Define the cluster connection settings in TCM.
2. Configure the cluster in TCM.
3. Start the cluster instances locally using the *tt* utility.

#### Defining the cluster's connection settings in TCM

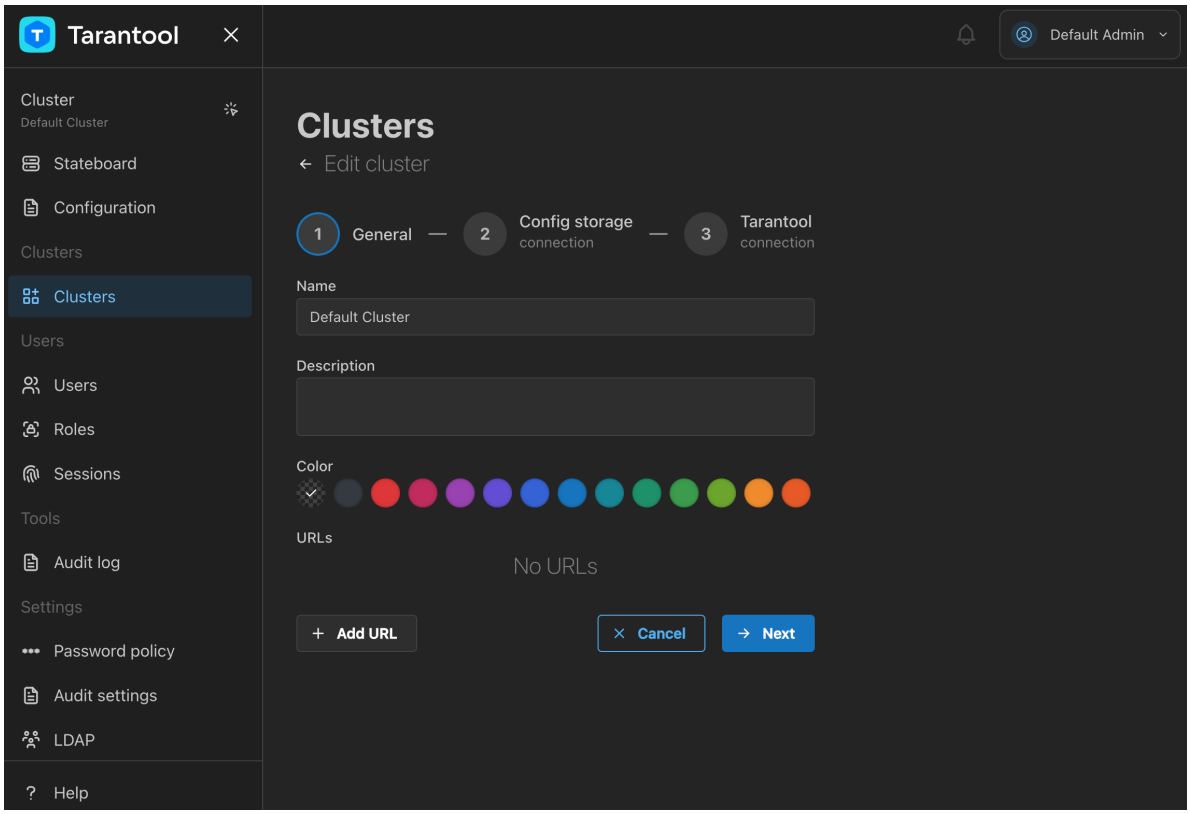
A freshly installed TCM has a predefined cluster named **Default cluster**. It doesn't have any configuration or topology out of the box. Its initial properties include the etcd and Tarantool connection parameters. Check these properties to find out where TCM sends the cluster configuration that you write.

To view the **Default cluster**'s properties:

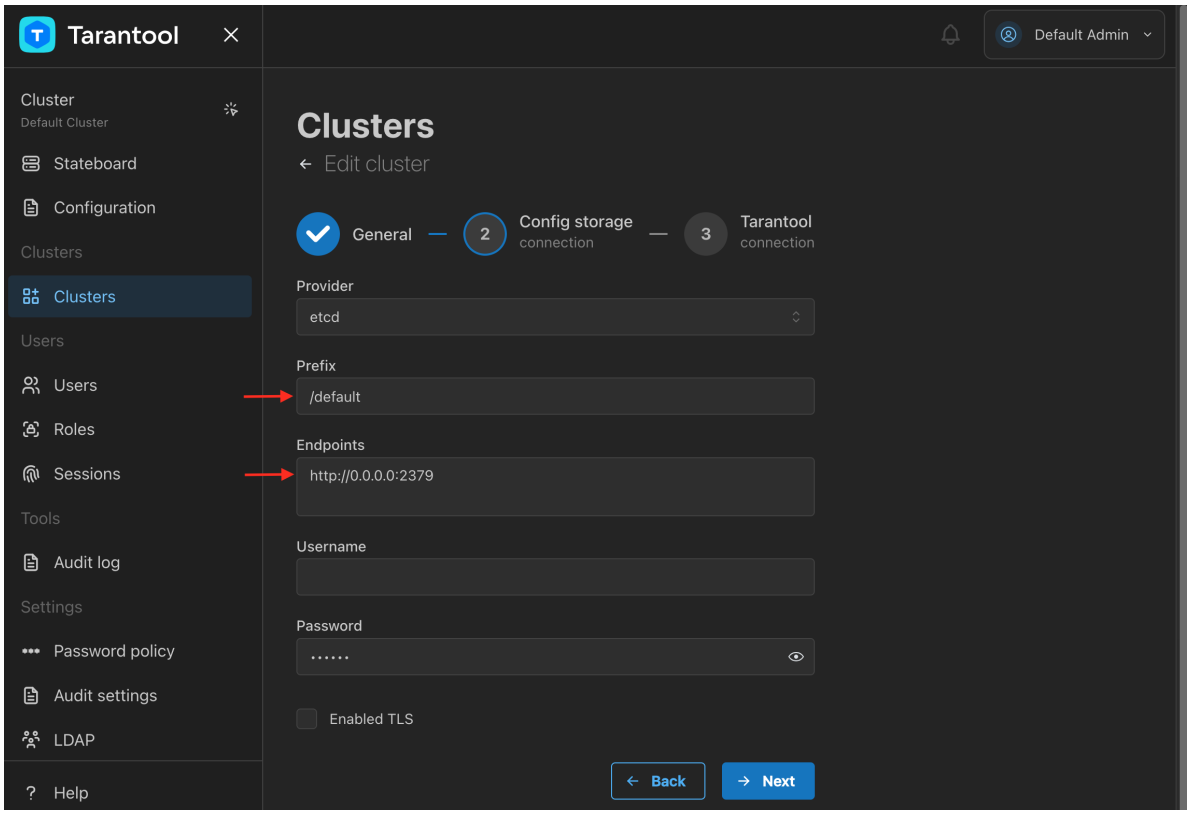
1. Go to **Clusters** and click **Edit** in the **Actions** menu opposite the cluster name.



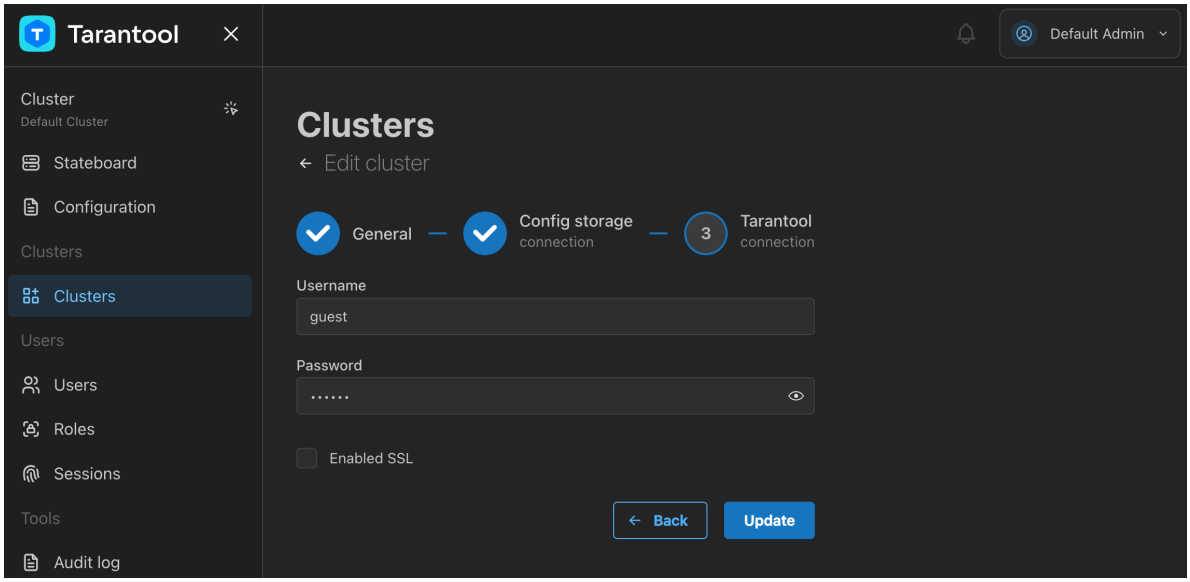

2. Click **Next** on the **General** tab.




3. Find the connection properties of the configuration storage that the cluster uses. By default, it's an etcd running on port `2379` (default etcd port) on the same host. The key prefix used for the cluster configuration is `/default`. Click **Next**.




4. Check the Tarantool user that TCM uses to connect to the cluster instances. It's `guest` by default.



### Configuring a cluster in TCM

TCM provides a web-based editor for writing cluster configurations. It is connected to the configuration storage (etcd in this case): all changes you make in the browser are sent to etcd in one click.

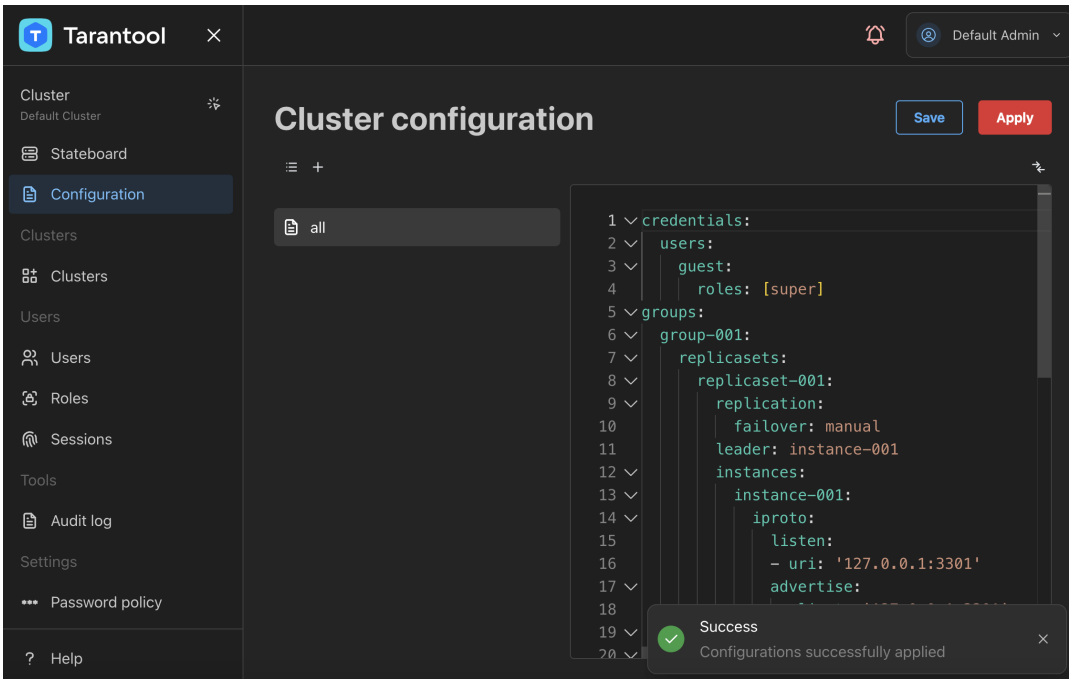
To write the cluster configuration and upload it to the etcd storage:

1. Go to **Configuration**.
2. Click + and provide an arbitrary name for the configuration file, for example, `all`.
3. Paste the following *YAML configuration* into the editor:

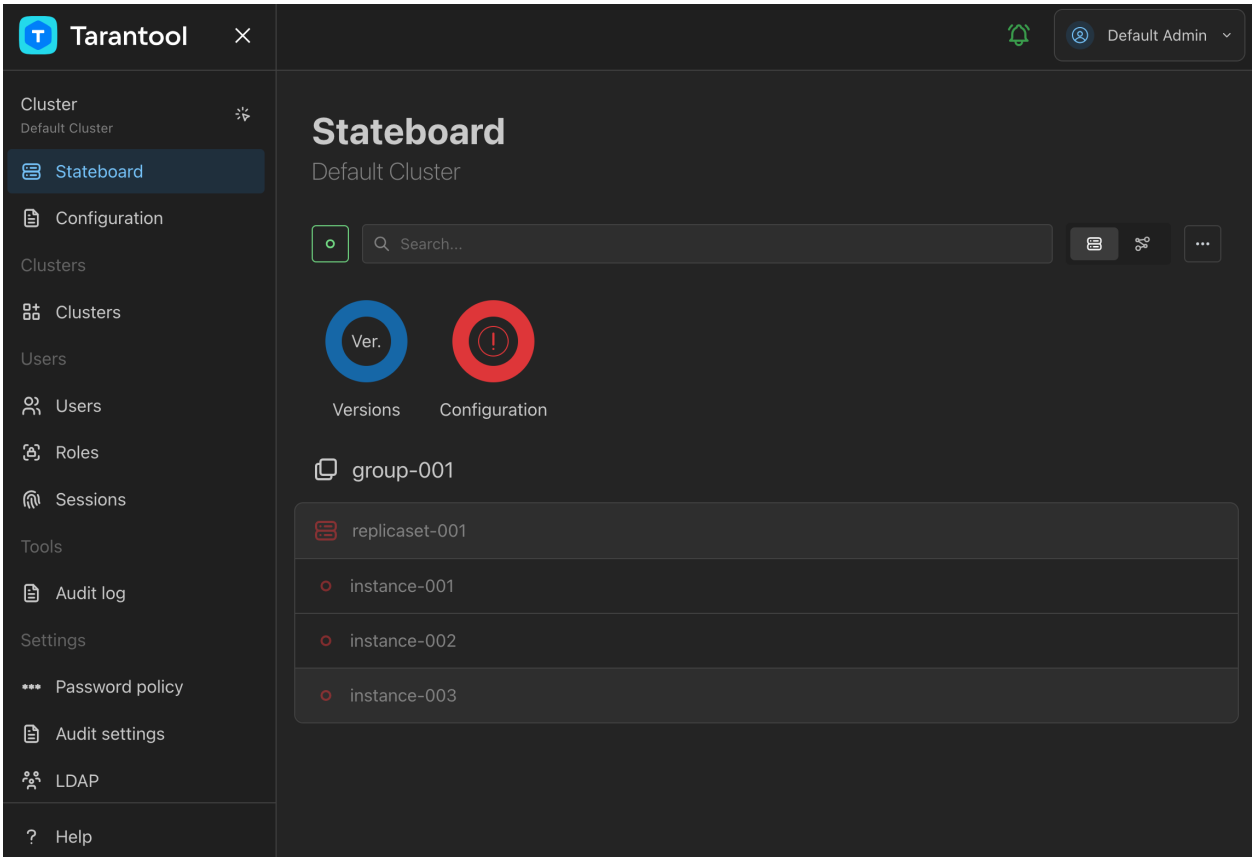
```
credentials:
  users:
    guest:
      roles: [super]
groups:
  group-001:
    replicasets:
      replicaset-001:
        replication:
          failover: manual
        leader: instance-001
        instances:
          instance-001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
              advertise:
                client: '127.0.0.1:3301'
          instance-002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
              advertise:
                client: '127.0.0.1:3302'
          instance-003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
              advertise:
                client: '127.0.0.1:3303'
```

   This configuration sets up a cluster of three nodes in one replica set: one leader and two followers.

4. Click **Apply** to send the configuration to etcd.




When the cluster configuration is saved, you can see the cluster topology on the **Stateboard** page:




However, the cluster instances are offline because they aren't deployed yet.

### Deploying the cluster locally

To deploy a local cluster based on the configuration from etcd:

1. Go to the system terminal you used when setting up Tarantool.
2. Create a new `tt` environment in a directory of your choice:

```
$ mkdir cluster-env
$ cd cluster-env/
$ tt init
```

3. Inside the `instances.enabled` directory of the created tt environment, create the `cluster` directory.

```
$ mkdir instances.enabled/cluster
$ cd instances.enabled/cluster/
```

4. Inside `instances.enabled/cluster`, create the `instances.yml` and `config.yaml` files:

   - `instances.yml` specifies instances to run in the current environment. In this example, there are three instances:

   ```
   instance-001:
   instance-002:
   instance-003:
   ```

   - `config.yaml` instructs `tt` to load the cluster configuration from etcd. The specified etcd location matches the configuration storage of the **Default cluster** in TCM:

   ```
   config:
     etcd:
       endpoints:
       - http://localhost:2379
       prefix: /default
   ```

5. Start the cluster from the `tt` environment root (the `cluster-env` directory):

```
$ tt start cluster
```

   To check how the cluster started, run `tt status`. This output should look like this:

```
$ tt status cluster
INSTANCE               STATUS      PID
cluster:instance-001     RUNNING     2058
cluster:instance-002     RUNNING     2059
cluster:instance-003     RUNNING     2060
```

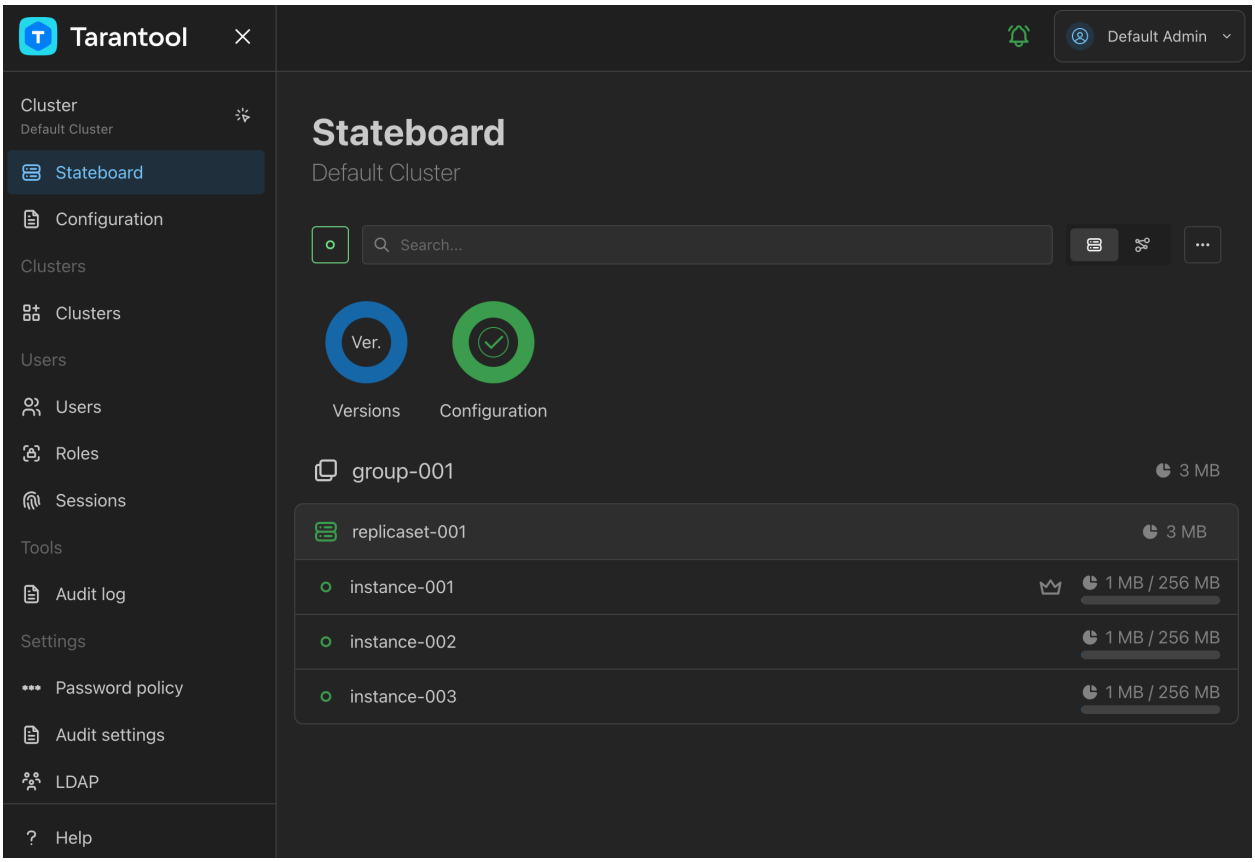
### 3.5.5 Managing the cluster in TCM

To learn to interact with a cluster in TCM, complete typical database tasks such as:

- Checking the cluster state.
- Creating a space.
- Writing data.
- Viewing data.

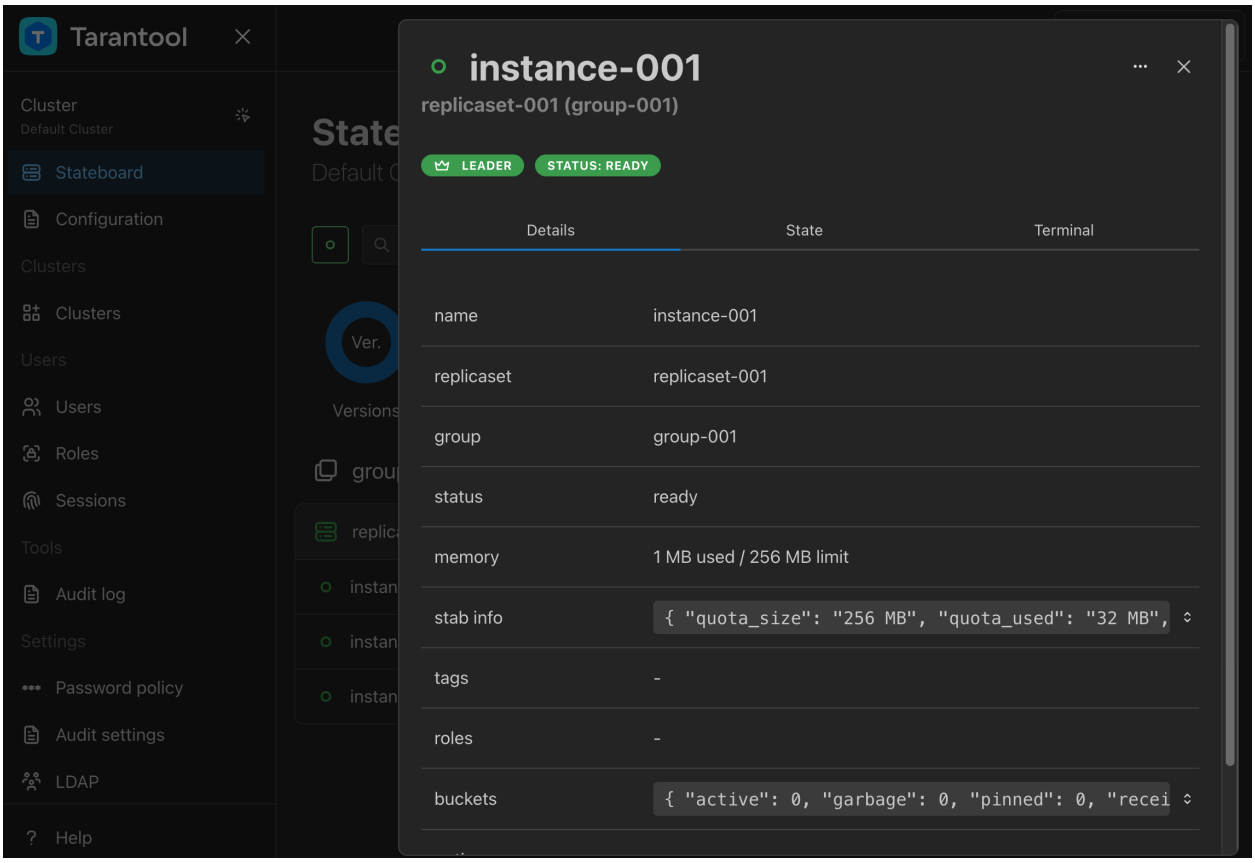
#### Checking cluster state

To check the cluster state in TCM, go to **Stateboard**. Here you see the overview of the cluster topology, health, memory consumption, and other information.

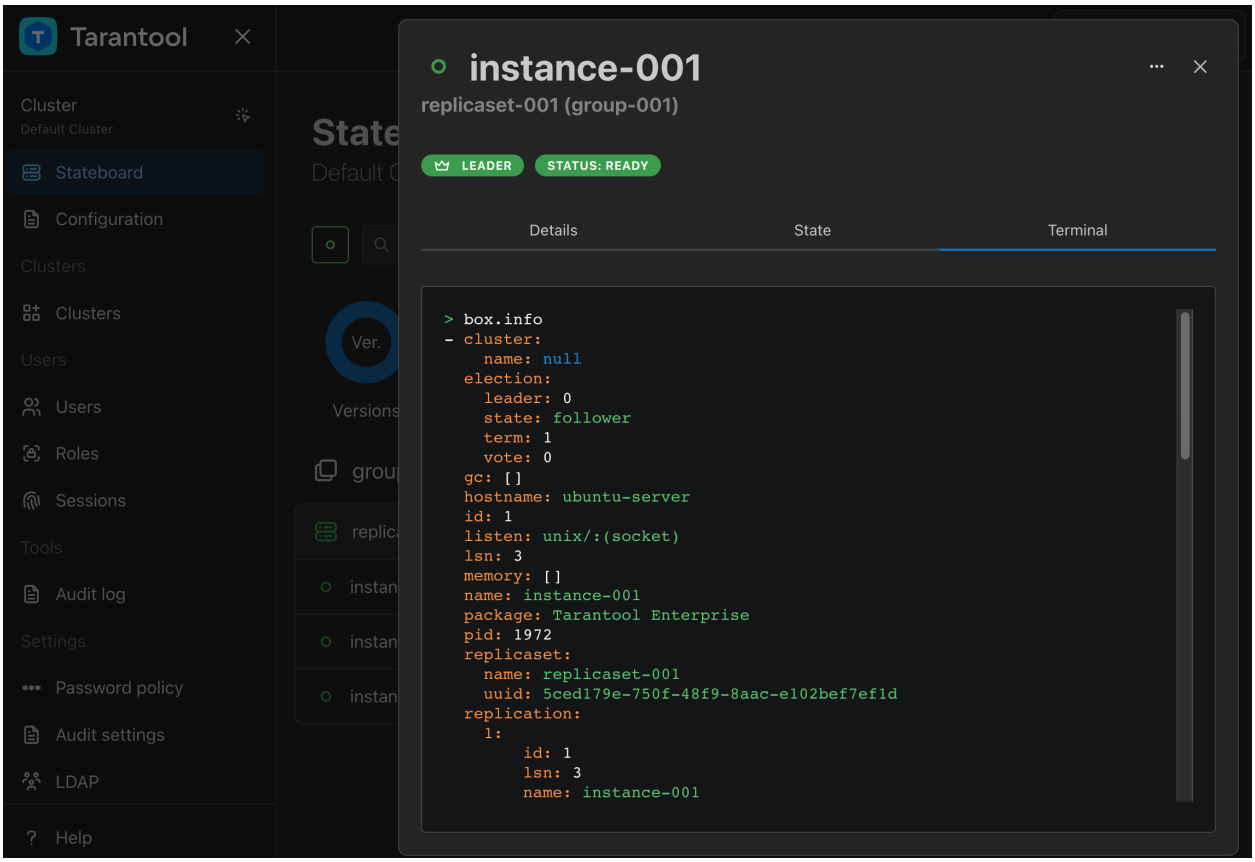



#### Connecting to an instance

To view detailed information about an instance, click its name in the instances list on the **Stateboard** page.




To connect to the instance interactively and execute code on it, go to the **Terminal** tab.




### Creating a space

Go to the terminal of `instance-001` (the leader instance) and run the following code to create a formatted space with a primary index in the cluster:

```
box.schema.space.create('bands')
box.space.bands:format({
    { name = 'id', type = 'unsigned' },
    { name = 'band_name', type = 'string' },
    { name = 'year', type = 'unsigned' }
})
box.space.bands:create_index('primary', { type = "tree", parts = { 'id' } })
```

### Writing data

Since `instance-001` is a read-write instance (its `box.info.ro` is `false`), the write requests must be executed on it. Run the following code in the `instance-001` terminal to write tuples in the space:

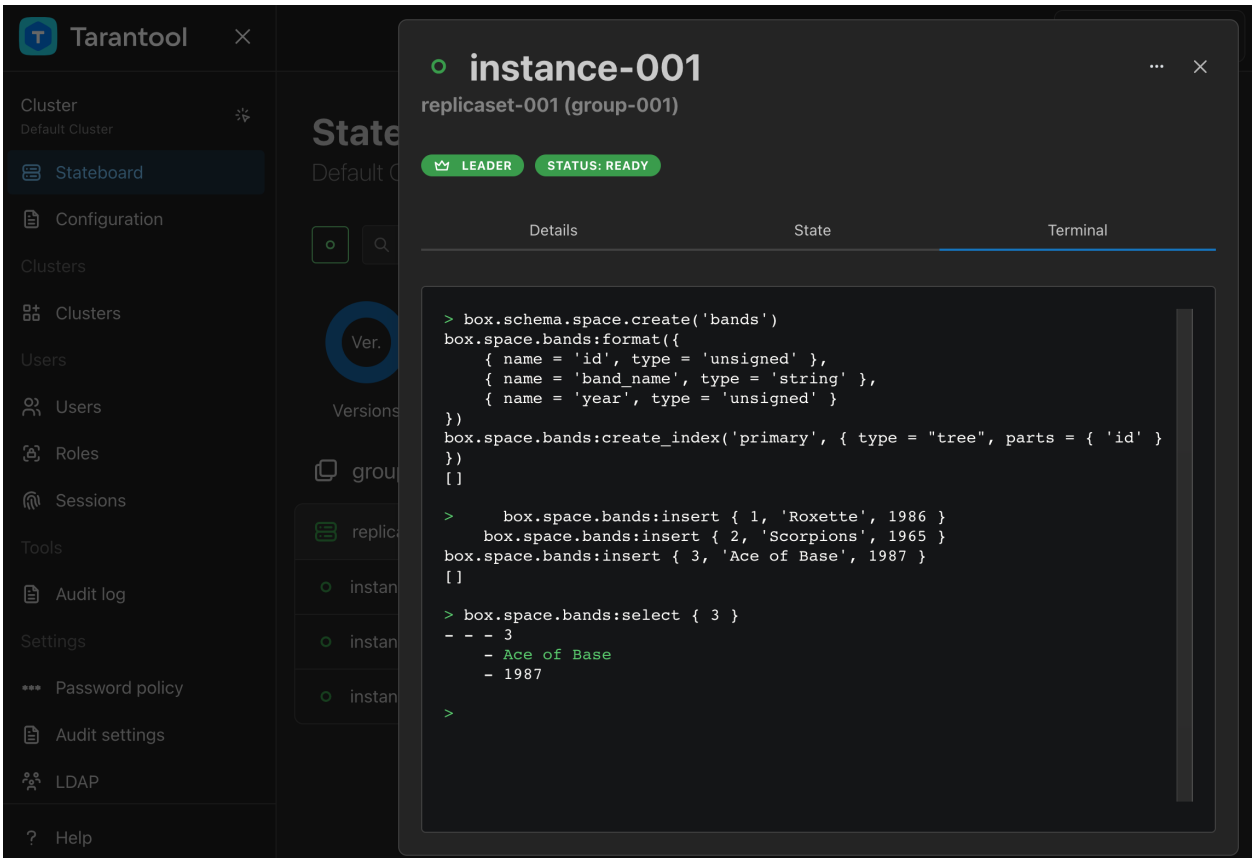
```
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }
```

### Reading data

Check the space's tuples by running a read request on `instance-001`:

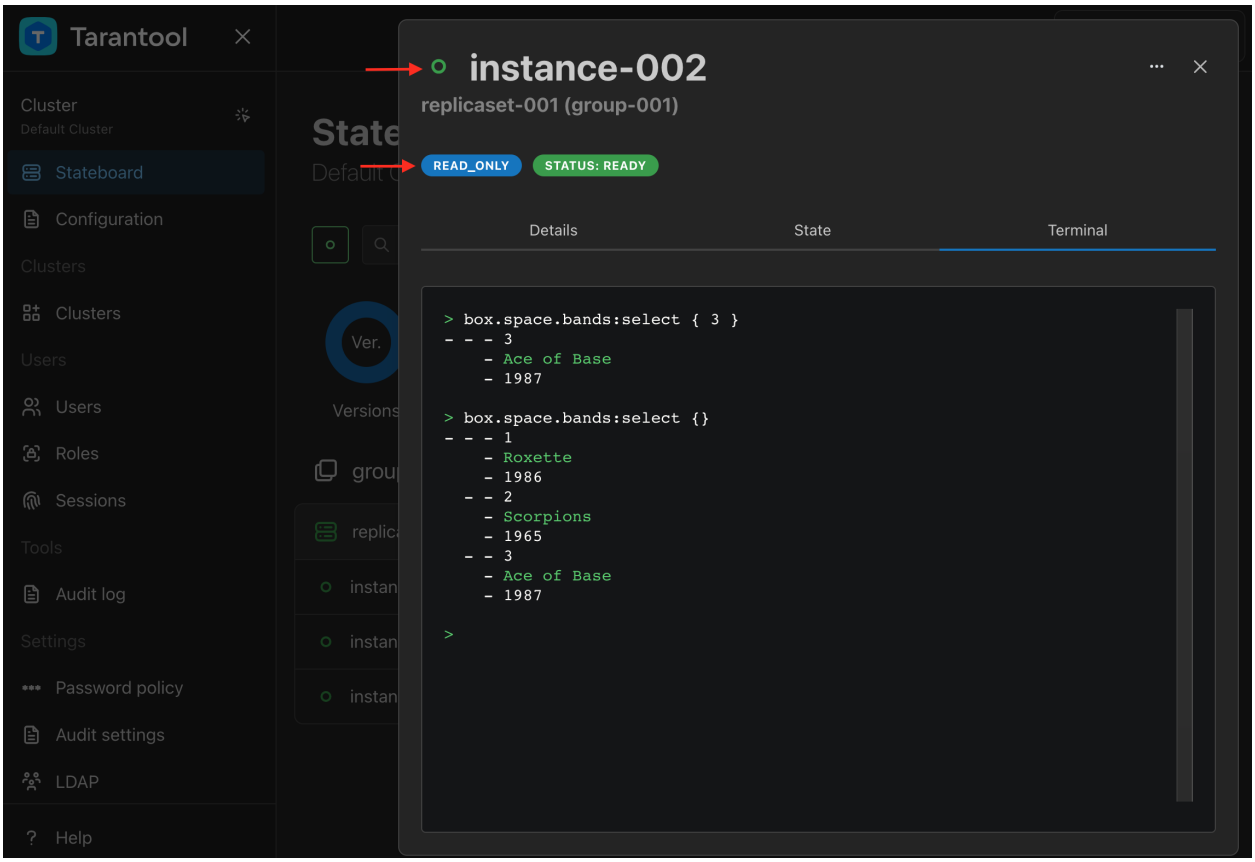
```
box.space.bands:select { 3 }
```

This is how it looks in TCM:




### Checking replication

To check that the data is replicated across instances, run the read request on any other instance – `instance-002` or `instance-003`. The result is the same as on `instance-001`.

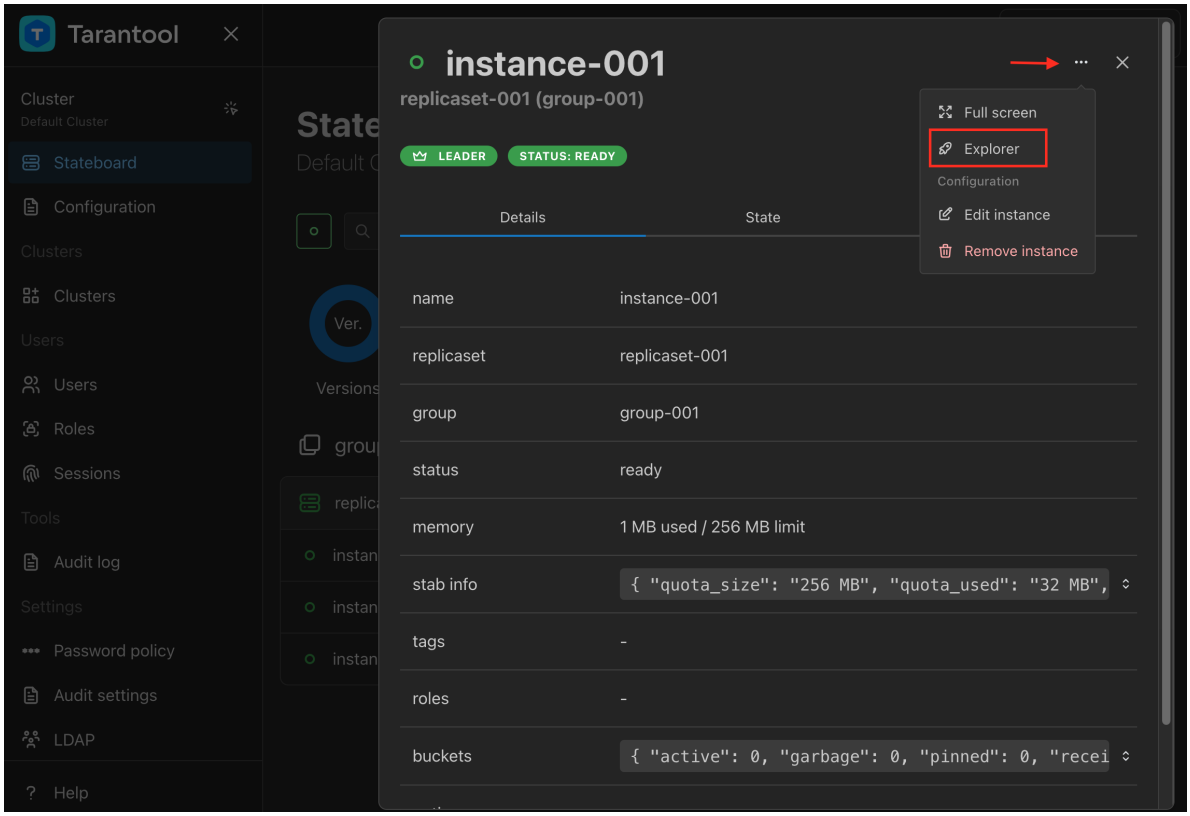



**Примечание:** If you try to execute a write request on any instance but `instance-001`, you get an error because these instances are configured to be read-only.

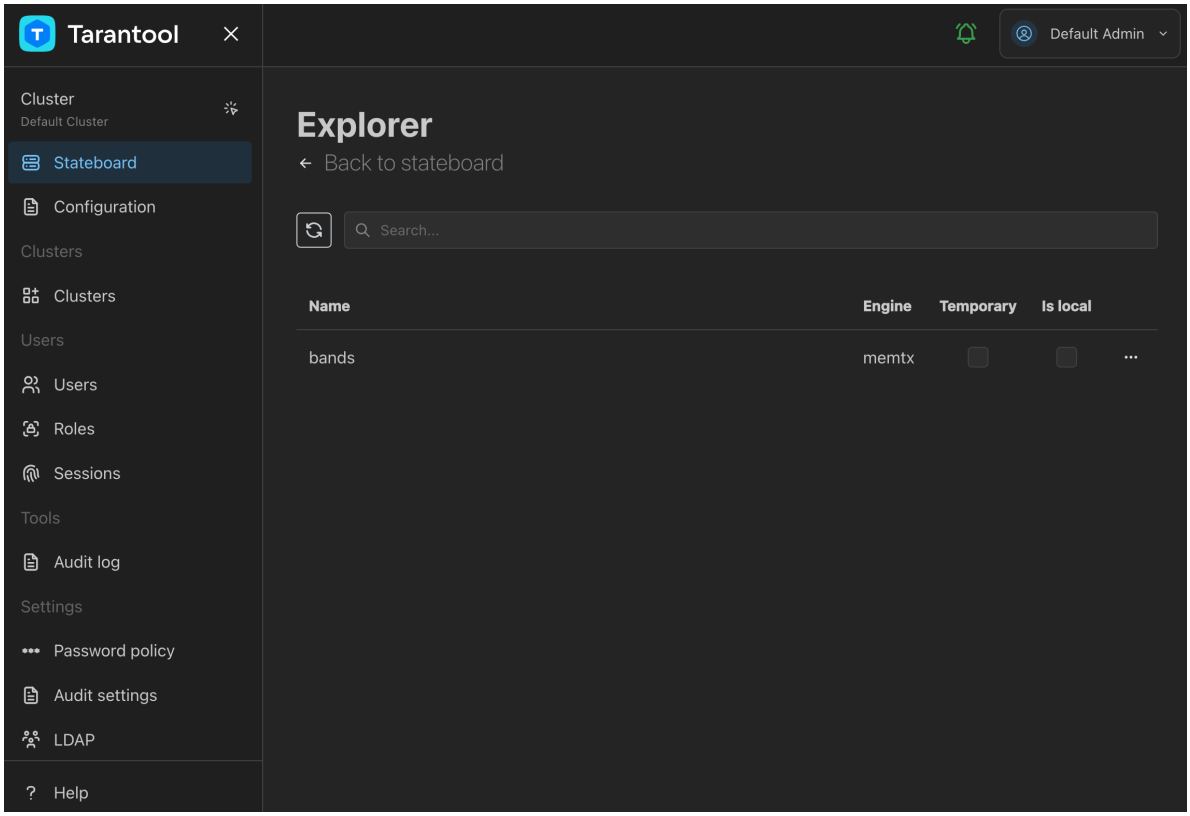
### Viewing data in TCM

TCM web UI includes a tool for viewing data stored in the cluster. To view the space tuples in TCM:

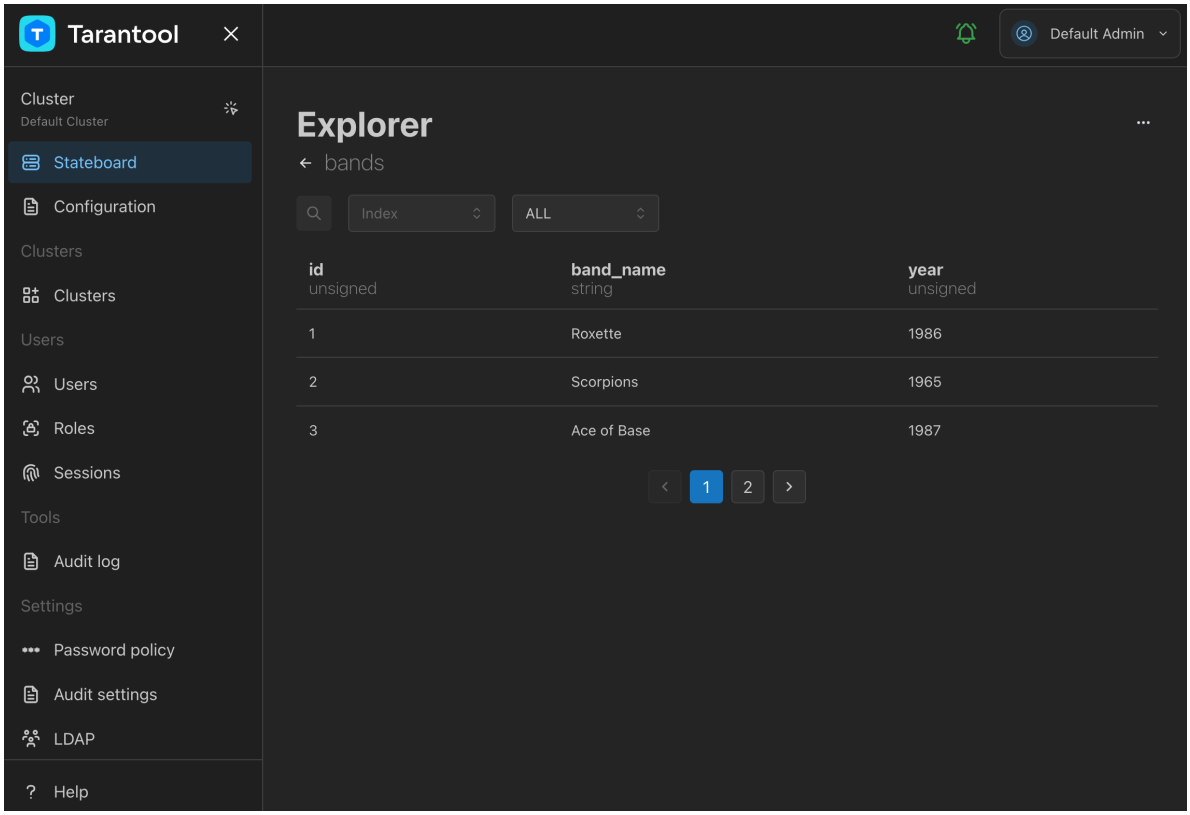
1. Click an instance name on the **Stateboard** page.
2. Open the **Actions** menu in the top-right corner and click **Explorer**.




This opens the page that lists user-created spaces on the instance.



3. Click **View** in the **Actions** menu of the space you want to see. The page shows all the tuples added previously.




## 3.6 Connecting to a database using net.box

**Examples on GitHub**: sample_db, net_box

The tutorial shows how to use `net.box` to connect to a remote Tarantool instance, perform CRUD operations, and execute stored procedures. For more information about the `net.box` module API, check *Модуль net.box*.

### 3.6.1 Sample database configuration

This section describes the *configuration* of a sample database that allows remote connections:

```
credentials:
  users:
    sampleuser:
      password: '123456'
      privileges:
      - permissions: [ read, write ]
        spaces: [ bands ]
      - permissions: [ execute ]
        functions: [ get_bands_older_than ]

groups:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'

app:
  file: 'myapp.lua'
```

- The configuration contains one instance that listens incoming requests on the `127.0.0.1:3301` address.
- `sampleuser` has *privileges* to select and modify data in the `bands` space and execute the `get_bands_older_than` stored function. This user can be used to connect to the instance remotely.
- `myapp.lua` defines how data is stored in a database and includes a stored function.

The `myapp.lua` file looks as follows:

```
-- Create a space --
box.schema.space.create('bands')

-- Specify field names and types --
box.space.bands:format({
    { name = 'id', type = 'unsigned' },
    { name = 'band_name', type = 'string' },
    { name = 'year', type = 'unsigned' }
})

-- Create indexes --
box.space.bands:create_index('primary', { parts = { 'id' } })
box.space.bands:create_index('band', { parts = { 'band_name' } })
box.space.bands:create_index('year_band', { parts = { { 'year' }, { 'band_name' } } })

-- Create a stored function --
box.schema.func.create('get_bands_older_than', {
    body = [[
    function(year)
        return box.space.bands.index.year_band:select({ year }, { iterator = 'LT', limit␣
↪= 10 })
    end
    ]]
})
```

You can find the full example on GitHub: sample_db.

### 3.6.2 Making net.box requests interactively

To try out `net.box` requests in the interactive console, start the *sample_db* application using `tt start`:

```
$ tt start sample_db
```

Then, use the *tt run -i* command to start an interactive console:

```
$ tt run -i
Tarantool 3.0.0-entrypoint-1144-geaff238d9
type 'help' for interactive help
tarantool>
```

In the console, you can create a `net.box` connection and try out data operations.

#### Создание подключения

To load the `net.box` module, use the `require()` directive:

```
net_box = require('net.box')
--[[
---
...
]]
```

To create a connection, pass a database URI to the *net_box.connect()* method:

```
conn = net_box.connect('sampleuser:123456@127.0.0.1:3301')
--[[
---
...
]]
```

*connection:ping()* can be used to check the connection status:

```
conn:ping()
--[[
---
- true
...
]]
```

#### Операции с данными

To get a space object and perform CRUD operations on it, use `conn.space.<space_name>`.

**Примечание:** Learn more about performing data operations from the *Примеры CRUD-операций* section.

### Inserting data

In the example below, four tuples are inserted into the `bands` space:

```
conn.space.bands:insert({ 1, 'Roxette', 1986 })
--[[
---
- - [1, 'Roxette', 1986]
...
]]
conn.space.bands:insert({ 2, 'Scorpions', 1965 })
--[[
---
- [2, 'Scorpions', 1965]
...
]]
conn.space.bands:insert({ 3, 'Ace of Base', 1987 })
--[[
---
- [3, 'Ace of Base', 1987]
...
]]
conn.space.bands:insert({ 4, 'The Beatles', 1960 })
--[[
---
- [4, 'The Beatles', 1960]
...
]]
```

### Querying data

The example below shows how to get a tuple by the specified primary key value:

```
conn.space.bands:select({ 1 })
--[[
---
- - [1, 'Roxette', 1986]
...
]]
```

You can also get a tuple by the value of the specified index as follows:

```
conn.space.bands.index.band:select({ 'The Beatles' })
--[[
---
- - [4, 'The Beatles', 1960]
...
]]
```

### Updating data

*space_object.update()* updates a tuple identified by the primary key. This method accepts a full key and an operation to execute:

```
conn.space.bands:update({ 2 }, { { '=', 'band_name', 'Pink Floyd' } })
--[[
---
- [2, 'Pink Floyd', 1965]
...
]]
```

*space_object.upsert()* updates an existing tuple or inserts a new one. In the example below, a new tuple is inserted:

```
conn.space.bands:upsert({ 5, 'The Rolling Stones', 1962 }, { { '=', 'band_name', 'The␣
↪Doors' } })
--[[
---
...
]]
```

In this example, *space_object.replace()* is used to delete the existing tuple and insert a new one:

```
conn.space.bands:replace({ 1, 'Queen', 1970 })
--[[
---
- [1, 'Queen', 1970]
...
]]
```

### Deleting data

The *space_object.delete()* call in the example below deletes a tuple whose primary key value is 5:

```
conn.space.bands:delete({ 5 })
--[[
---
- [5, 'The Rolling Stones', 1962]
...
]]
```

### Executing stored procedures

To execute a stored procedure, use the *connection:call()* method:

```
conn:call('get_bands_older_than', { 1966 })
-- ---
-- - [[2, 'Pink Floyd', 1965], [4, 'The Beatles', 1960]]
-- ...
```

#### Закрытие подключения

The *connection:close()* method can be used to close the connection when it is no longer needed:

```
conn:close()
--[[
---
...
]]
```

**Примечание:** You can find the example with all the requests above on GitHub: net_box.

## 3.7 Подключаемся к базе из разных языков программирования

В *предыдущих разделах* вы узнали, как создать базу данных Tarantool. Теперь давайте посмотрим, как подключаться к базе данных из разных языков программирования, таких как Python, PHP, Go и C++, и выполнять типовые запросы для работы с данными (select, insert, delete и так далее).

### 3.7.1 Подключение из Python

#### Подготовка

Перед тем как идти дальше, выполним следующие действия:

1. Установим библиотеку `tarantool`. Рекомендуется использовать `python3` и `pip3`.
2. *Запустим* Tarantool (локально или в Docker) и обязательно создадим базу данных с тестовыми данными, как показано в *предыдущем разделе*:

```
box.cfg{listen = 3301}
s = box.schema.space.create('tester')
s:format({
         {name = 'id', type = 'unsigned'},
         {name = 'band_name', type = 'string'},
         {name = 'year', type = 'unsigned'}
         })
s:create_index('primary', {
         type = 'hash',
         parts = {'id'}
         })
s:create_index('secondary', {
         type = 'hash',
         parts = {'band_name'}
         })
s:insert{1, 'Roxette', 1986}
s:insert{2, 'Scorpions', 2015}
s:insert{3, 'Ace of Base', 1993}
```

**Важно:** Не закрывайте окно терминала с запущенным Tarantool – оно пригодится нам позднее.

3. Чтобы иметь возможность подключаться к Tarantool в качестве администратора, сменим пароль пользователя `admin`:

```
box.schema.user.passwd('pass')
```

### Подключение к Tarantool

Для подключения к серверу достаточно выполнить следующее:

```
>>> import tarantool
>>> connection = tarantool.connect("localhost", 3301)
```

Также при необходимости можно указать имя пользователя и пароль:

```
>>> tarantool.connect("localhost", 3301, user=username, password=password)
```

По умолчанию используется пользователь `guest`.

### Работа с данными

*Спейс* — это контейнер для *кортежей*. Чтобы обратиться к спейсу как к именованному объекту, воспользуемся функцией `connection.space`:

```
>>> tester = connection.space('tester')
```

### Вставка данных

Для вставки нового кортежа в спейс воспользуемся функцией `insert`:

```
>>> tester.insert((4, 'ABBA', 1972))
[4, 'ABBA', 1972]
```

### Получение данных

Сначала выберем кортеж по первичному ключу (в нашем примере первичный индекс —это индекс `primary`, построенный по полю `id` в каждом кортеже). Воспользуемся функцией `select`:

```
>>> tester.select(4)
[4, 'ABBA', 1972]
```

Теперь поищем кортежи по вторичному ключу. Для этого нужно указать номер *или* имя вторичного индекса.

Сначала сделаем запрос по номеру индекса:

```
>>> tester.select('Scorpions', index=1)
[2, 'Scorpions', 2015]
```

(Мы указываем `index=1`, потому что индексы в Tarantool нумеруются с нуля, а в данном случае мы обращаемся к индексу, который создавали вторым.)

Теперь сделаем аналогичный запрос по имени индекса и получим тот же результат:

```
>>> tester.select('Scorpions', index='secondary')
[2, 'Scorpions', 2015]
```

А чтобы выбрать все кортежи из спейса, вызовем `select` без аргументов:

```
>>> tester.select()
```

### Обновление данных

Обновим значение поля с помощью `update`:

```
>>> tester.update(4, [('=', 1, 'New group'), ('+', 2, 2)])
```

Здесь мы обновляем значение поля `1` и увеличиваем значение поля `2` для кортежа с `id = 4`. Если кортежа с таким `id` нет, то Tarantool вернет ошибку.

Теперь с помощью функции `replace` мы полностью заменим кортеж с совпадающим первичным ключом. Если кортежа с указанным первичным ключом не существует, то эта операция ни к чему не приведет.

```
>>> tester.replace((4, 'New band', 2015))
```

Также мы можем обновлять данные с помощью функции `upsert`, которая работает аналогично `update`, но создает новый кортеж, если старый не был найден.

```
>>> tester.upsert((4, 'Another band', 2000), [('+', 2, 5)])
```

Здесь мы увеличиваем на 5 значение поля `2` в кортеже с `id = 4`, — или же вставляем кортеж `(4, "Another band", 2000)`, если такого нет.

### Удаление данных

Чтобы удалить кортеж, нужно использовать `delete(primary_key)`:

```
>>> tester.delete(4)
[4, 'New group', 2012]
```

Для удаления всех кортежей в спейсе (или всего спейса целиком) нужно воспользоваться функцией `call`. Мы поговорим о ней подробнее в *следующем разделе*.

Чтобы удалить все кортежи в спейсе, нужно вызвать функцию `space:truncate`:

```
>>> connection.call('box.space.tester:truncate', ())
```

Чтобы удалить весь спейс, нужно вызвать функцию `space:drop`. Для выполнения следующей команды необходимо подключиться из-под пользователя `admin`:

```
>>> connection.call('box.space.tester:drop', ())
```

#### Исполнение хранимых процедур

Перейдем в терминал с запущенным Tarantool.

---

**Примечание:** О том, как установить удаленное подключение к Tarantool, можно прочитать здесь:

- как подключиться к Tarantool, запущенному локально
- как подключиться к Tarantool, запущенному в Docker-контейнере

---

Напишем простую функцию на Lua:

```
function sum(a, b)
    return a + b
end
```

Итак, теперь у нас есть функция, описанная в Tarantool. Чтобы вызвать ее из `python`, нам нужна функция `call`:

```
>>> connection.call('sum', (3, 2))
5
```

Также мы можем передать на выполнение любой Lua-код. Для этого воспользуемся функцией `eval`:

```
>>> connection.eval('return 4 + 5')
9
```

#### Сравнение функций

Чтобы подробнее ознакомиться со всеми доступными Python-коннекторами, посмотрите *таблицу сравнения их функций* в разделе «Коннекторы».

### 3.7.2 Подключение из PHP

#### Подготовка

Перед тем как идти дальше, выполним следующие действия:

1. Установим библиотеку `tarantool/client`.
2. *Запустим* Tarantool (локально или в Docker) и обязательно создадим базу данных с тестовыми данными, как показано в *предыдущем разделе*:

```
box.cfg{listen = 3301}
s = box.schema.space.create('tester')
s:format({
         {name = 'id', type = 'unsigned'},
         {name = 'band_name', type = 'string'},
         {name = 'year', type = 'unsigned'}
         })
s:create_index('primary', {
         type = 'hash',
         parts = {'id'}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        })
s:create_index('secondary', {
        type = 'hash',
        parts = {'band_name'}
        })
s:insert{1, 'Roxette', 1986}
s:insert{2, 'Scorpions', 2015}
s:insert{3, 'Ace of Base', 1993}
```

**Важно:** Не закрывайте окно терминала с запущенным Tarantool – оно пригодится нам позднее.

3. Чтобы иметь возможность подключаться к Tarantool в качестве администратора, сменим пароль пользователя `admin`:

```
box.schema.user.passwd('pass')
```

### Подключение к Tarantool

Для настройки подключения к серверу достаточно выполнить следующее:

```
use Tarantool\Client\Client;

require __DIR__.'/vendor/autoload.php';
$client = Client::fromDefaults();
```

Само подключение будет установлено при первом запросе. Также при необходимости можно указать имя пользователя и пароль:

```
$client = Client::fromOptions([
    'uri' => 'tcp://127.0.0.1:3301',
    'username' => '<username>',
    'password' => '<password>'
]);
```

По умолчанию используется пользователь `guest`.

### Работа с данными

*Спейс* — это контейнер для *кортежей*. Чтобы обратиться к спейсу как к именованному объекту, воспользуйтесь функцией `getSpace`:

```
$tester = $client->getSpace('tester');
```

#### Вставка данных

Для вставки нового кортежа в спейс воспользуемся функцией `insert`:

```
$result = $tester->insert([4, 'ABBA', 1972]);
```

#### Получение данных

Сначала выберем кортеж по первичному ключу (в нашем примере первичный индекс —это индекс `primary`, построенный по полю `id` в каждом кортеже). Воспользуемся функцией `select`:

```
use Tarantool\Client\Schema\Criteria;

$result = $tester->select(Criteria::key([4]));
printf(json_encode($result));
```

```
[[4, 'ABBA', 1972]]
```

Теперь поищем кортежи по вторичному ключу. Для этого нужно указать номер *или* имя вторичного индекса.

Сначала сделаем запрос по номеру индекса:

```
$result = $tester->select(Criteria::index(1)->andKey(['Scorpions']));
printf(json_encode($result));
```

```
[2, 'Scorpions', 2015]
```

(Мы указываем `index(1)`, потому что индексы в Tarantool нумеруются с нуля, а в данном случае мы обращаемся к индексу, который создавали вторым.)

Теперь сделаем аналогичный запрос по имени индекса и получим тот же результат:

```
$result = $tester->select(Criteria::index('secondary')->andKey(['Scorpions']));
printf(json_encode($result));
```

```
[2, 'Scorpions', 2015]
```

А чтобы выбрать все кортежи из спейса, вызовем `select`:

```
$result = $tester->select(Criteria::allIterator());
```

#### Обновление данных

Обновите значение поля с помощью `update`:

```
use Tarantool\Client\Schema\Operations;

$result = $tester->update([4], Operations::set(1, 'New group')->andAdd(2, 2));
```

Здесь обновляется значение поля 1 и увеличивается значение поля 2 для кортежа с `id = 4`. Если кортежа с таким `id` нет, то Tarantool вернет ошибку.

Теперь с помощью функции `replace` мы полностью заменим кортеж с совпадающим первичным ключом. Если кортежа с указанным первичным ключом не существует, то эта операция ни к чему не приведет.

```
$result = $tester->replace([4, 'New band', 2015]);
```

Также мы можем обновлять данные с помощью функции `upsert`, которая работает аналогично `update`, но создает новый кортеж, если старый не был найден.

```
use Tarantool\Client\Schema\Operations;

$tester->upsert([4, 'Another band', 2000], Operations::add(2, 5));
```

Здесь значение поля 2 в кортеже с `id = 4` увеличится на 5, — или же произойдет вставка кортежа `(4, "Another band", 2000)`, если такого нет.

### Удаление данных

Чтобы удалить кортеж, нужно использовать `delete(primary_key)`:

```
$result = $tester->delete([4]);
```

Для удаления всех кортежей в спейсе (или всего спейса целиком) нужно воспользоваться функцией `call`. Мы поговорим о ней подробнее в *следующем разделе*.

Чтобы удалить все кортежи в спейсе, нужно вызвать функцию `space:truncate`:

```
$result = $client->call('box.space.tester:truncate');
```

Чтобы удалить весь спейс, нужно вызвать функцию `space:drop`. Для выполнения следующей команды необходимо подключиться из-под пользователя `admin`:

```
$result = $client->call('box.space.tester:drop');
```

### Исполнение хранимых процедур

Перейдем в терминал с запущенным Tarantool.

---

**Примечание:** О том, как установить удаленное подключение к Tarantool, можно прочитать здесь:

- как подключиться к Tarantool, запущенному локально
- как подключиться к Tarantool, запущенному в Docker-контейнере

---

Напишем простую функцию на Lua:

```
function sum(a, b)
    return a + b
end
```

Итак, теперь у нас есть функция, описанная в Tarantool. Чтобы вызвать ее из `php`, нам нужна функция `call`:

```
$result = $client->call('sum', 3, 2);
```

Также мы можем передать на выполнение любой Lua-код. Для этого воспользуемся функцией `eval`:

```
$result = $client->evaluate('return 4 + 5');
```

### 3.7.3 Подключение из Go

#### Подготовка

Перед тем как идти дальше, выполним следующие действия:

1. Установим библиотеку `go-tarantool`.
2. *Запустим* Tarantool (локально или в Docker) и обязательно создадим базу данных с тестовыми данными, как показано в *предыдущем разделе*:

```
box.cfg{listen = 3301}
s = box.schema.space.create('tester')
s:format({
         {name = 'id', type = 'unsigned'},
         {name = 'band_name', type = 'string'},
         {name = 'year', type = 'unsigned'}
         })
s:create_index('primary', {
         type = 'hash',
         parts = {'id'}
         })
s:create_index('secondary', {
         type = 'hash',
         parts = {'band_name'}
         })
s:insert{1, 'Roxette', 1986}
s:insert{2, 'Scorpions', 2015}
s:insert{3, 'Ace of Base', 1993}
```

---

**Важно:** Не закрывайте окно терминала с запущенным Tarantool – оно пригодится нам позднее.

---

3. Чтобы иметь возможность подключаться к Tarantool в качестве администратора, сменим пароль пользователя `admin`:

```
box.schema.user.passwd('pass')
```

### Подключение к Tarantool

Простая программа, выполняющая подключение к серверу, будет выглядеть так:

```
package main

import (
    "fmt"

    "github.com/tarantool/go-tarantool"
)

func main() {

    conn, err := tarantool.Connect("127.0.0.1:3301", tarantool.Opts{
            User: "admin",
            Pass: "pass",
    })

    if err != nil {
            log.Fatalf("Connection refused")
    }

    defer conn.Close()

    // Ваш код общения с базой

}
```

По умолчанию используется пользователь `guest`.

### Работа с данными

#### Вставка данных

Для вставки нового кортежа в *спейс* воспользуйтесь функцией `Insert`:

```
resp, err = conn.Insert("tester", []interface{}{4, "ABBA", 1972})
```

В этом примере в спейс `tester` вставляется кортеж `(4, "ABBA", 1972)`.

Код ответа и данные можно получить из структуры tarantool.Response:

```
code := resp.Code
data := resp.Data
```

### Получение данных

Чтобы выбрать кортеж из спейса, воспользуемся функцией Select:

```
resp, err = conn.Select("tester", "primary", 0, 1, tarantool.IterEq, []interface{}{4})
```

В этом примере выполняется поиск кортежа по первичному ключу с `offset = 0` и `limit = 1` в спейсе `tester` (первичный индекс в нашем примере – это индекс `primary`, построенный по полю `id` в каждом кортеже).

Теперь поищем по вторичному ключу:

```
resp, err = conn.Select("tester", "secondary", 0, 1, tarantool.IterEq, []interface{}{
↪"ABBA"})
```

Наконец, было бы интересно сделать полную выборку даных из спейса. Но в рамках языка Go эта задача не решается в одну строчку. *Пример* такой программы вы можете посмотреть в отдельном разделе документации.

Более сложные примеры выборок можно увидеть тут: https://github.com/tarantool/go-tarantool#usage

### Обновление данных

Обновим значение поля с помощью `Update`:

```
resp, err = conn.Update("tester", "primary", []interface{}{4}, []interface{}{[]interface
↪{}{"+", 2, 3}})
```

Здесь значение поля 2 для кортежа с `id = 4` мы увеличиваем на 3. Если кортежа с таким `id` нет, то Tarantool вернет ошибку.

Теперь с помощью функции `Replace` мы полностью заменим кортеж с совпадающим первичным ключом. Если кортежа с указанным первичным ключом не существует, то эта операция ни к чему не приведет.

```
resp, err = conn.Replace("tester", []interface{}{4, "New band", 2011})
```

Также мы можем обновлять данные с помощью функции `Upsert`, которая работает аналогично `Update`, но создает новый кортеж, если старый не был найден.

```
resp, err = conn.Upsert("tester", []interface{}{4, "Another band", 2000}, []interface{}
↪{[]interface{}{"+", 2, 5}})
```

Здесь значение третьего поля в кортеже с `id = 4` мы увеличиваем на 5, — или же вставляем кортеж `(4, "Another band", 2000)`, если такого нет.

### Удаление данных

To delete a tuple, use `connection.Delete`:

```
resp, err = conn.Delete("tester", "primary", []interface{}{4})
```

Для удаления всех кортежей в спейсе (или всего спейса целиком), нужно воспользоваться функцией `Call`. Мы поговорим о ней подробнее в *следующем разделе*.

Чтобы удалить все кортежи в спейсе, нужно вызвать функцию `space:truncate`:

```
resp, err = conn.Call("box.space.tester:truncate", []interface{}{})
```

Чтобы удалить весь спейс, нужно вызвать функцию `space:drop`. Для выполнения следующей команды необходимо подключиться из-под пользователя `admin`:

```
resp, err = conn.Call("box.space.tester:drop", []interface{}{})
```

### Исполнение хранимых процедур

Перейдем в терминал с запущенным Tarantool.

---

**Примечание:** О том, как установить удаленное подключение к Tarantool, можно прочитать здесь:

- как подключиться к Tarantool, запущенному локально
- как подключиться к Tarantool, запущенному в Docker-контейнере

---

Напишем простую функцию на Lua:

```
function sum(a, b)
    return a + b
end
```

Итак, теперь у нас есть функция, описанная в Tarantool. Чтобы вызвать ее из `go`, нам нужна функция `Call`:

```
resp, err = conn.Call("sum", []interface{}{2, 3})
```

Также мы можем передать на выполнение любой Lua-код. Для этого воспользуемся функцией `Eval`:

```
resp, err = connection.Eval("return 4 + 5", []interface{}{})
```

### Сравнение функций

Есть еще два доступных коннектора от опенсорс-сообщества:

- viciious/go-tarantool
- FZambia/tarantool.

Чтобы подробнее ознакомиться с этими коннекторами, посмотрите *таблицу сравнения функций* всех доступных Go-коннекторов.

### 3.7.4 Connecting to Tarantool from C++

To simplify the start of your working with the Tarantool C++ connector, we will use the example application from the connector repository. We will go step by step through the application code and explain what each part does.

The following main topics are discussed in this manual:


- *Pre-requisites*
- *Connecting to Tarantool*
- *Working with requests*
- *Building and launching C++ application*
- *Decoding and reading the data*


#### Pre-requisites

To go through this Getting Started exercise, you need the following pre-requisites to be done:

- *clone the connector source code and ensure having Tarantool and third-party software*
- *start Tarantool and create a database*
- *set up access rights*.

#### Installation

The Tarantool C++ connector is currently supported for Linux only.

The connector itself is a header-only library, so, it doesn't require installation and building as such. All you need is to clone the connector source code and *embed* it in your C++ project.

Also, make sure you have other necessary software and Tarantool installed.

1. Make sure you have the following third-party software. If you miss some of the items, install them:
   - Git, a version control system
   - unzip utility
   - gcc compiler complied with the C++17 standard
   - cmake and make tools.
2. If you don't have Tarantool on your OS, install it in one of the ways:
   - from a package–refer to OS-specific instructions
   - from the source.
3. Clone the Tarantool C++ connector repository.

```
git clone git@github.com:tarantool/tntcxx.git
```

### Starting Tarantool and creating a database

Start Tarantool locally or in Docker and create a space with the following schema and index:

```
box.cfg{listen = 3301}
t = box.schema.space.create('t')
t:format({
         {name = 'id', type = 'unsigned'},
         {name = 'a', type = 'string'},
         {name = 'b', type = 'number'}
         })
t:create_index('primary', {
         type = 'hash',
         parts = {'id'}
         })
```

**Важно:** Do not close the terminal window where Tarantool is running. You will need it later to connect to Tarantool from your C++ application.

### Setting up access rights

To be able to execute the necessary operations in Tarantool, you need to grant the `guest` user with the read-write rights. The simplest way is to grant the user with the *super role*:

```
box.schema.user.grant('guest', 'super')
```

### Connecting to Tarantool

There are three main parts of the C++ connector: the IO-zero-copy buffer, the msgpack encoder/decoder, and the client that handles requests.

To set up connection to a Tarantool instance from a C++ application, you need to do the following:

- *embed the connector into the application*
- *instantiate a connector client and a connection object*
- *define connection parameters and invoke the method to connect*
- *define error handling behavior*.

### Embedding connector

Embed the connector in your C++ application by including the main header:

```
#include "../src/Client/Connector.hpp"
```

### Instantiating objects

First, we should create a connector client. It can handle many connections to Tarantool instances asynchronously. To instantiate a client, you should specify the buffer and the network provider implementations as template parameters. The connector's main class has the following signature:

```
template<class BUFFER, class NetProvider = EpollNetProvider<BUFFER>>
class Connector;
```

The buffer is parametrized by allocator. It means that users can choose which allocator will be used to provide memory for the buffer's blocks. Data is organized into a linked list of blocks of fixed size that is specified as the template parameter of the buffer.

You can either implement your own buffer or network provider or use the default ones as we do in our example. So, the default connector instantiation looks as follows:

```
using Buf_t = tnt::Buffer<16 * 1024>;
#include "../src/Client/LibevNetProvider.hpp"
using Net_t = LibevNetProvider<Buf_t, DefaultStream>;
```

```
Connector<Buf_t, Net_t> client;
```

To use the `BUFFER` class, the buffer header should also be included:

```
#include "../src/Buffer/Buffer.hpp"
```

A client itself is not enough to work with Tarantool instances–we also need to create connection objects. A connection also takes the buffer and the network provider as template parameters. Note that they must be the same as ones of the client:

```
Connection<Buf_t, Net_t> conn(client);
```

### Connecting

Our *Tarantool instance* is listening to the `3301` port on `localhost`. Let's define the corresponding variables as well as the `WAIT_TIMEOUT` variable for connection timeout.

```
const char *address = "127.0.0.1";
int port = 3301;
int WAIT_TIMEOUT = 1000; //milliseconds
```

To connect to the Tarantool instance, we should invoke the `Connector::connect()` method of the client object and pass three arguments: connection instance, address, and port.

```
int rc = client.connect(conn, {.address = address,
                               .service = std::to_string(port),
                               /*.user = ...,*/
                               /*.passwd = ...,*/
                               /* .transport = STREAM_SSL, */});
```

### Error handling

Implementation of the connector is exception free, so we rely on the return codes: in case of fail, the `connect()` method returns `rc < 0`. To get the error message corresponding to the last error occured during communication with the instance, we can invoke the `Connection::getError()` method.

```
if (rc != 0) {
        //assert(conn.getError().saved_errno != 0);
        std::cerr << conn.getError().msg << std::endl;
        return -1;
}
```

To reset connection after errors, that is, to clean up the error message and connection status, the `Connection::reset()` method is used.

### Working with requests

In this section, we will show how to:

- *prepare different types of requests*
- *send the requests*
- *receive and handle responses*.

We will also go through the *case of having several connections* and executing a number of requests from different connections simultaneously.

In our example C++ application, we execute the following types of requests:

- `ping`
- `replace`
- `select`.

---

**Примечание:** Examples on other request types, namely, `insert`, `delete`, `upsert`, and `update`, will be added to this manual later.

---

Each request method returns a request ID that is a sort of future. This ID can be used to get the response message when it is ready. Requests are queued in the output buffer of connection until the `Connector::wait()` method is called.

### Preparing requests

At this step, requests are encoded in the MessagePack format and saved in the output connection buffer. They are ready to be sent but the network communication itself will be done later.

Let's remind that for the requests manipulating with data we are dealing with the Tarantool space `t` *created earlier*, and the space has the following format:

```
t:format({
         {name = 'id', type = 'unsigned'},
         {name = 'a', type = 'string'},
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        {name = 'b', type = 'number'}
        })
```

**ping**

```
rid_t ping = conn.ping();
```

**replace**

Equals to Lua request `<space_name>:replace(pk_value, "111", 1)`.

```
uint32_t space_id = 512;
int pk_value = 666;
std::tuple data = std::make_tuple(pk_value /* field 1*/, "111" /* field 2*/, 1.01 /*␣
↪field 3*/);
rid_t replace = conn.space[space_id].replace(data);
```

**select**

Equals to Lua request `<space_name>.index[0]:select({pk_value}, {limit = 1})`.

```
uint32_t index_id = 0;
uint32_t limit = 1;
uint32_t offset = 0;
IteratorType iter = IteratorType::EQ;
auto i = conn.space[space_id].index[index_id];
rid_t select = i.select(std::make_tuple(pk_value), limit, offset, iter);
```

### Sending requests

To send requests to the server side, invoke the `client.wait()` method.

```
client.wait(conn, ping, WAIT_TIMEOUT);
```

The `wait()` method takes the connection to poll, the request ID, and, optionally, the timeout as parameters. Once a response for the specified request is ready, `wait()` terminates. It also provides a negative return code in case of system related fails, for example, a broken or timeouted connection. If `wait()` returns `0`, then a response has been received and expected to be parsed.

Now let's send our requests to the Tarantool instance. The `futureIsReady()` function checks availability of a future and returns `true` or `false`.

```
while (! conn.futureIsReady(ping)) {
        /*
         * wait() is the main function responsible for sending/receiving
         * requests and implements event-loop under the hood. It may
         * fail due to several reasons:
         *  - connection is timed out;
         *  - connection is broken (e.g. closed);
         *  - epoll is failed.
         */
        if (client.wait(conn, ping, WAIT_TIMEOUT) != 0) {
                std::cerr << conn.getError().msg << std::endl;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                conn.reset();
        }
}
```

### Receiving responses

To get the response when it is ready, use the `Connection::getResponse()` method. It takes the request ID and returns an optional object containing the response. If the response is not ready yet, the method returns `std::nullopt`. Note that on each future, `getResponse()` can be called only once: it erases the request ID from the internal map once it is returned to a user.

A response consists of a header and a body (`response.header` and `response.body`). Depending on success of the request execution on the server side, body may contain either runtime error(s) accessible by `response.body.error_stack` or data (tuples)–`response.body.data`. In turn, data is a vector of tuples. However, tuples are not decoded and come in the form of pointers to the start and the end of msgpacks. See the *«Decoding and reading the data»* section to understand how to decode tuples.

There are two options for single connection it regards to receiving responses: we can either wait for one specific future or for all of them at once. We'll try both options in our example. For the `ping` request, let's use the first option.

```
std::optional<Response<Buf_t>> response = conn.getResponse(ping);
/*
 * Since conn.futureIsReady(ping) returned <true>, then response
 * must be ready.
 */
assert(response != std::nullopt);
/*
 * If request is successfully executed on server side, response
 * will contain data (i.e. tuple being replaced in case of :replace()
 * request or tuples satisfying search conditions in case of :select();
 * responses for pings contain nothing - empty map).
 * To tell responses containing data from error responses, one can
 * rely on response code storing in the header or check
 * Response->body.data and Response->body.error_stack members.
 */
printResponse<Buf_t>(*response);
```

For the `replace` and `select` requests, let's examine the option of waiting for both futures at once.

```
/* Let's wait for both futures at once. */
std::vector<rid_t> futures(2);
futures[0] = replace;
futures[1] = select;
/* No specified timeout means that we poll futures until they are ready.*/
client.waitAll(conn, futures);
for (size_t i = 0; i < futures.size(); ++i) {
        assert(conn.futureIsReady(futures[i]));
        response = conn.getResponse(futures[i]);
        assert(response != std::nullopt);
        printResponse<Buf_t>(*response);
}
```

### Several connections at once

Now, let's have a look at the case when we establish two connections to Tarantool instance simultaneously.

```
/* Let's create another connection. */
Connection<Buf_t, Net_t> another(client);
if (client.connect(another, {.address = address,
                             .service = std::to_string(port),
                             /* .transport = STREAM_SSL, */}) != 0) {
        std::cerr << conn.getError().msg << std::endl;
        return -1;
}
/* Simultaneously execute two requests from different connections. */
rid_t f1 = conn.ping();
rid_t f2 = another.ping();
/*
 * waitAny() returns the first connection received response.
 * All connections registered via :connect() call are participating.
 */
std::optional<Connection<Buf_t, Net_t>> conn_opt = client.waitAny(WAIT_TIMEOUT);
Connection<Buf_t, Net_t> first = *conn_opt;
if (first == conn) {
        assert(conn.futureIsReady(f1));
        (void) f1;
} else {
        assert(another.futureIsReady(f2));
        (void) f2;
}
```

### Closing connections

Finally, a user is responsible for closing connections.

```
client.close(conn);
client.close(another);
```

### Building and launching C++ application

Now, we are going to build our example C++ application, launch it to connect to the Tarantool instance and execute all the requests defined.

Make sure you are in the root directory of the cloned C++ connector repository. To build the example application:

```
cd examples
cmake .
make
```

Make sure the *Tarantool session* you started earlier is running. Launch the application:

```
./Simple
```

As you can see from the execution log, all the connections to Tarantool defined in our application have been established and all the requests have been executed successfully.

### Decoding and reading the data

Responses from a Tarantool instance contain raw data, that is, the data encoded into the MessagePack tuples. To decode client's data, the user has to write their own decoders (readers) based on the database schema and include them in one's application:

```
#include "Reader.hpp"
```

To show the logic of decoding a response, we will use the reader from our example.

First, the structure corresponding our *example space format* is defined:

```
/**
 * Corresponds to tuples stored in user's space:
 * box.execute("CREATE TABLE t (id UNSIGNED PRIMARY KEY, a TEXT, d DOUBLE);")
 */
struct UserTuple {
        uint64_t field1;
        std::string field2;
        double field3;

        static constexpr auto mpp = std::make_tuple(
                &UserTuple::field1, &UserTuple::field2, &UserTuple::field3);
};
```

### Base reader prototype

Prototype of the base reader is given in `src/mpp/Dec.hpp`:

```
template <class BUFFER, Type TYPE>
struct SimpleReaderBase : DefaultErrorHandler {
   using BufferIterator_t = typename BUFFER::iterator;
   /* Allowed type of values to be parsed. */
   static constexpr Type VALID_TYPES = TYPE;
   BufferIterator_t* StoreEndIterator() { return nullptr; }
};
```

Every new reader should inherit from it or directly from the `DefaultErrorHandler`.

### Parsing values

To parse a particular value, we should define the `Value()` method. First two arguments of the method are common and unused as a rule, but the third one defines the parsed value. In case of POD (Plain Old Data) structures, it's enough to provide a byte-to-byte copy. Since there are fields of three different types in our schema, let's define the corresponding `Value()` functions:

#### Parsing array

It's also important to understand that a tuple itself is wrapped in an array, so, in fact, we should parse the array first. Let's define another reader for that purpose.

#### Setting reader

The `SetReader()` method sets the reader that is invoked while each of the array's entries is parsed. To make two readers defined above work, we should create a decoder, set its iterator to the position of the encoded tuple, and invoke the `Read()` method (the code block below is from the example application).

## 3.8 Developing applications with Tarantool

Using Tarantool as an application server, you can write your own applications. Tarantool's native language for writing applications is Lua, so a typical application would be a file that contains your Lua script. But you can also write applications in C or C++.

### 3.8.1 Launching an application

Using Tarantool as an application server, you can write your own applications. Tarantool's native language for writing applications is Lua, so a typical application would be a file that contains your Lua script. But you can also write applications in C or C++.

---

**Примечание:** If you're new to Lua, we recommend going over the interactive Tarantool tutorial before proceeding with this chapter. To launch the tutorial, say `tutorial()` in Tarantool console:

```
tarantool> tutorial()
---
- |
 Tutorial -- Screen #1 -- Hello, Moon
 ====================================

 Welcome to the Tarantool tutorial.
 It will introduce you to Tarantool's Lua application server
 and database server, which is what's running what you're seeing.
 This is INTERACTIVE -- you're expected to enter requests
 based on the suggestions or examples in the screen's text.
 <...>
```

---

Let's create and launch our first Lua application for Tarantool. Here's a simplest Lua application, the good old «Hello, world!»:

```
#!/usr/bin/env tarantool
print('Hello, world!')
```

We save it in a file. Let it be `myapp.lua` in the current directory.

Now let's discuss how we can launch our application with Tarantool.

### Launching in Docker

If we run Tarantool in a Docker container, the following command will start Tarantool without any application:

```
$ # create a temporary container and run it in interactive mode
$ docker run --rm -t -i tarantool/tarantool:latest
```

To run Tarantool with our application, we can say:

```
$ # create a temporary container and
$ # launch Tarantool with our application
$ docker run --rm -t -i \
             -v `pwd`/myapp.lua:/opt/tarantool/myapp.lua \
             -v /data/dir/on/host:/var/lib/tarantool \
             tarantool/tarantool:latest tarantool /opt/tarantool/myapp.lua
```

Here two resources on the host get mounted in the container:

- our application file (myapp.lua) and
- Tarantool data directory (`/data/dir/on/host`).

By convention, the directory for Tarantool application code inside a container is `/opt/tarantool`, and the directory for data is `/var/lib/tarantool`.

### Launching a binary program

If we run Tarantool from a package or from a *source build*, we can launch our application:

- in the script mode,
- as a server application, or
- as a daemon service.

The simplest way is to pass the filename to Tarantool at start:

```
$ tarantool myapp.lua
Hello, world!
$
```

Tarantool starts, executes our script in the **script mode** and exits.

Now let's turn this script into a **server application**. We use *box.cfg* from Tarantool's built-in Lua module to:

- launch the database (a database has a persistent on-disk state, which needs to be restored after we start an application) and
- configure Tarantool as a server that accepts requests over a TCP port.

We also add some simple database logic, using *space.create()* and *create_index()* to create a space with a primary index. We use the function *box.once()* to make sure that our logic will be executed only once when the database is initialized for the first time, so we don't try to create an existing space or index on each invocation of the script:

```
#!/usr/bin/env tarantool
-- Configure database
box.cfg {
   listen = 3301
}
box.once("bootstrap", function()
   box.schema.space.create('tweedledum')
   box.space.tweedledum:create_index('primary',
       { type = 'TREE', parts = {1, 'unsigned'}})
end)
```

Now we launch our application in the same manner as before:

```
$ tarantool myapp.lua
Hello, world!
2017-08-11 16:07:14.250 [41436] main/101/myapp.lua C> version 2.1.0-429-g4e5231702
2017-08-11 16:07:14.250 [41436] main/101/myapp.lua C> log level 5
2017-08-11 16:07:14.251 [41436] main/101/myapp.lua I> mapping 1073741824 bytes for tuple␣
↪arena...
2017-08-11 16:07:14.255 [41436] main/101/myapp.lua I> recovery start
2017-08-11 16:07:14.255 [41436] main/101/myapp.lua I> recovering from `./
↪00000000000000000000.snap'
2017-08-11 16:07:14.271 [41436] main/101/myapp.lua I> recover from `./
↪00000000000000000000.xlog'
2017-08-11 16:07:14.271 [41436] main/101/myapp.lua I> done `./00000000000000000000.xlog'
2017-08-11 16:07:14.272 [41436] main/102/hot_standby I> recover from `./
↪00000000000000000000.xlog'
2017-08-11 16:07:14.274 [41436] iproto/102/iproto I> binary: started
2017-08-11 16:07:14.275 [41436] iproto/102/iproto I> binary: bound to [::]:3301
2017-08-11 16:07:14.275 [41436] main/101/myapp.lua I> done `./00000000000000000000.xlog'
2017-08-11 16:07:14.278 [41436] main/101/myapp.lua I> ready to accept requests
```

This time, Tarantool executes our script and keeps working as a server, accepting TCP requests on port 3301. We can see Tarantool in the current session's process list:

```
$ ps | grep "tarantool"
  PID TTY           TIME CMD
41608 ttys001       0:00.47 tarantool myapp.lua <running>
```

But the Tarantool instance will stop if we close the current terminal window. To detach Tarantool and our application from the terminal window, we can launch it in the **daemon mode**. To do so, we add some parameters to `box.cfg{}`:

- *background* = `true` that actually tells Tarantool to work as a daemon service,
- *log* = `'dir-name'` that tells the Tarantool daemon where to store its log file (other log settings are available in Tarantool *log* module), and
- *pid_file* = `'file-name'` that tells the Tarantool daemon where to store its pid file.

For example:

```
box.cfg {
   listen = 3301,
   background = true,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    log = '1.log',
    pid_file = '1.pid'
}
```

We launch our application in the same manner as before:

```
$ tarantool myapp.lua
Hello, world!
$
```

Tarantool executes our script, gets detached from the current shell session (you won't see it with `ps | grep "tarantool"`) and continues working in the background as a daemon attached to the global session (with SID = 0):

```
$ ps -ef | grep "tarantool"
  PID SID     TIME  CMD
42178   0  0:00.72 tarantool myapp.lua <running>
```

Now that we have discussed how to create and launch a Lua application for Tarantool, let's dive deeper into programming practices.

### 3.8.2 Создание приложения

Further we walk you through key programming practices that will give you a good start in writing Lua applications for Tarantool. We will implement a real microservice based on Tarantool! It is a backend for a simplified version of Pokémon Go, a location-based augmented reality game launched in mid-2016.

In this game, players use the GPS capability of a mobile device to locate, catch, battle, and train virtual monsters called «pokémon» that appear on the screen as if they were in the same real-world location as the player.

To stay within the walk-through format, let's narrow the original gameplay as follows. We have a map with pokémon spawn locations. Next, we have multiple players who can send catch-a-pokémon requests to the server (which runs our Tarantool microservice). The server responds whether the pokémon is caught or not, increases the player's pokémon counter if yes, and triggers the respawn-a-pokémon method that spawns a new pokémon at the same location in a while.

We leave client-side applications outside the scope of this story. However, we promise a mini-demo in the end to simulate real users and give us some fun.

Follow these topics to implement our application:

#### Modules, rocks and applications

To make our game logic available to other developers and Lua applications, let's put it into a Lua module.

A **module** (called «rock» in Lua) is an optional library which enhances Tarantool functionality. So, we can install our logic as a module in Tarantool and use it from any Tarantool application or module. Like applications, modules in Tarantool can be written in Lua (rocks), C or C++.

Modules are good for two things:

- easier **code management** (reuse, packaging, versioning), and

- hot **code reload** without restarting the Tarantool instance.

Technically, a module is a file with source code that exports its functions in an API. For example, here is a Lua module named `mymodule.lua` that exports one function named `myfun`:

```
local exports = {}
exports.myfun = function(input_string)
   print('Hello', input_string)
end
return exports
```

To launch the function `myfun()` – from another module, from a Lua application, or from Tarantool itself, – we need to save this module as a file, then load this module with the `require()` directive and call the exported function.

For example, here's a Lua application that uses `myfun()` function from `mymodule.lua` module:

```
-- loading the module
local mymodule = require('mymodule')

-- calling myfun() from within test() function
local test = function()
  mymodule.myfun()
end
```

A thing to remember here is that the `require()` directive takes load paths to Lua modules from the `package.path` variable. This is a semicolon-separated string, where a question mark is used to interpolate the module name. By default, this variable contains system-wide Lua paths and the working directory. But if we put our modules inside a specific folder (e.g. `scripts/`), we need to add this folder to `package.path` before any calls to `require()`:

```
package.path = 'scripts/?.lua;' .. package.path
```

For our microservice, a simple and convenient solution would be to put all methods in a Lua module (say `pokemon.lua`) and to write a Lua application (say `game.lua`) that initializes the gaming environment and starts the game loop.

Now let's get down to implementation details. In our game, we need three entities:

- **map**, which is an array of pokémons with coordinates of respawn locations; in this version of the game, let a location be a rectangle identified with two points, upper-left and lower-right;
- **player**, which has an ID, a name, and coordinates of the player's location point;
- **pokémon**, which has the same fields as the player, plus a status (active/inactive, that is present on the map or not) and a catch probability (well, let's give our pokémons a chance to escape :-) )

We'll store these entities as tuples in Tarantool spaces. But to deliver our backend application as a microservice, the good practice would be to send/receive our data in the universal JSON format, thus using Tarantool as a document storage.

### Avro schemas

To store JSON data as tuples, we will apply a savvy practice which reduces data footprint and ensures all stored documents are valid. We will use Tarantool module avro-schema which checks the schema of a JSON document and converts it to a Tarantool tuple. The tuple will contain only field values, and thus take a lot less space than the original document. In avro-schema terms, converting JSON documents to tuples is «flattening», and restoring the original documents is «unflattening».

First you need to *install* the module with `tt rocks install avro-schema`.

Further usage is quite straightforward:

(1) For each entity, we need to define a schema in Apache Avro schema syntax, where we list the entity's fields with their names and Avro data types.

(2) At initialization, we call `avro-schema.create()` that creates objects in memory for all schema entities, and `compile()` that generates flatten/unflatten methods for each entity.

(3) Further on, we just call flatten/unflatten methods for a respective entity on receiving/sending the entity's data.

Here's what our schema definitions for the player and pokémon entities look like:

```
local schema = {
    player = {
        type="record",
        name="player_schema",
        fields={
            {name="id", type="long"},
            {name="name", type="string"},
            {
                name="location",
                type= {
                    type="record",
                    name="player_location",
                    fields={
                        {name="x", type="double"},
                        {name="y", type="double"}
                    }
                }
            }
        }
    },
    pokemon = {
        type="record",
        name="pokemon_schema",
        fields={
            {name="id", type="long"},
            {name="status", type="string"},
            {name="name", type="string"},
            {name="chance", type="double"},
            {
                name="location",
                type= {
                    type="record",
                    name="pokemon_location",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                    fields={
                        {name="x", type="double"},
                        {name="y", type="double"}
                    }
                }
            }
        }
    }
}
```

And here's how we create and compile our entities at initialization:

```
-- load avro-schema module with require()
local avro = require('avro_schema')

-- create models
local ok_m, pokemon = avro.create(schema.pokemon)
local ok_p, player = avro.create(schema.player)
if ok_m and ok_p then
    -- compile models
    local ok_cm, compiled_pokemon = avro.compile(pokemon)
    local ok_cp, compiled_player = avro.compile(player)
    if ok_cm and ok_cp then
        -- start the game
        <...>
    else
        log.error('Schema compilation failed')
    end
else
    log.info('Schema creation failed')
end
return false
```

As for the map entity, it would be an overkill to introduce a schema for it, because we have only one map in the game, it has very few fields, and – which is most important – we use the map only inside our logic, never exposing it to external users.

Next, we need methods to implement the game logic. To simulate object-oriented programming in our Lua code, let's store all Lua functions and shared variables in a single local variable (let's name it as `game`). This will allow us to address functions or variables from within our module as `self.func_name` or `self.var_name`. Like this:

```
local game = {
    -- a local variable
    num_players = 0,

    -- a method that prints a local variable
    hello = function(self)
      print('Hello! Your player number is ' .. self.num_players .. '.')
    end,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    -- a method that calls another method and returns a local variable
    sign_in = function(self)
      self.num_players = self.num_players + 1
      self:hello()
      return self.num_players
    end
}
```

In OOP terms, we can now regard local variables inside `game` as object fields, and local functions as object methods.

---

**Примечание:** In this manual, Lua examples use **local** variables. Use **global** variables with caution, since the module's users may be unaware of them.

To enable/disable the use of undeclared global variables in your Lua code, use Tarantool's *strict* module.

---

So, our game module will have the following methods:

- `catch()` to calculate whether the pokémon was caught (besides the coordinates of both the player and pokémon, this method will apply a probability factor, so not every pokémon within the player's reach will be caught);
- `respawn()` to add missing pokémons to the map, say, every 60 seconds (we assume that a frightened pokémon runs away, so we remove a pokémon from the map on any catch attempt and add it back to the map in a while);
- `notify()` to log information about caught pokémons (like «Player 1 caught pokémon A»);
- `start()` to initialize the game (it will create database spaces, create and compile avro schemas, and launch `respawn()`).

Besides, it would be convenient to have methods for working with Tarantool storage. For example:

- `add_pokemon()` to add a pokémon to the database, and
- `map()` to populate the map with all pokémons stored in Tarantool.

We'll need these two methods primarily when initializing our game, but we can also call them later, for example to test our code.

### Bootstrapping a database

Let's discuss game initialization. In `start()` method, we need to populate Tarantool spaces with pokémon data. Why not keep all game data in memory? Why use a database? The answer is: *persistence*. Without a database, we risk losing data on power outage, for example. But if we store our data in an in-memory database, Tarantool takes care to persist it on disk whenever it's changed. This gives us one more benefit: quick startup in case of failure. Tarantool has a *smart algorithm* that quickly loads all data from disk into memory on startup, so the warm-up takes little time.

We'll be using functions from Tarantool built-in *box* module:

- `box.schema.create_space('pokemons')` to create a space named `pokemon` for storing information about pokémons (we don't create a similar space for players, because we intend to only send/receive player information via API calls, so we needn't store it);
- `box.space.pokemons:create_index('primary', {type = 'hash', parts = {1, 'unsigned'}})` to create a primary HASH index by pokémon ID;

- `box.space.pokemons:create_index('status', {type = 'tree', parts = {2, 'str'}})` to create a secondary TREE index by pokémon status.

Notice the `parts =` argument in the index specification. The pokémon ID is the first field in a Tarantool tuple since it's the first member of the respective Avro type. So does the pokémon status. The actual JSON document may have ID or status fields at any position of the JSON map.

The implementation of `start()` method looks like this:

```
-- create game object
start = function(self)
    -- create spaces and indexes
    box.once('init', function()
        box.schema.create_space('pokemons')
        box.space.pokemons:create_index(
            "primary", {type = 'hash', parts = {1, 'unsigned'}}
        )
        box.space.pokemons:create_index(
            "status", {type = "tree", parts = {2, 'str'}}
        )
    end)

    -- create models
    local ok_m, pokemon = avro.create(schema.pokemon)
    local ok_p, player = avro.create(schema.player)
    if ok_m and ok_p then
        -- compile models
        local ok_cm, compiled_pokemon = avro.compile(pokemon)
        local ok_cp, compiled_player = avro.compile(player)
        if ok_cm and ok_cp then
            -- start the game
            <...>
        else
            log.error('Schema compilation failed')
        end
    else
        log.info('Schema creation failed')
    end
    return false
end
```

### GIS

Now let's discuss `catch()`, which is the main method in our gaming logic.

Here we receive the player's coordinates and the target pokémon's ID number, and we need to answer whether the player has actually caught the pokémon or not (remember that each pokémon has a chance to escape).

First thing, we validate the received player data against its *Avro schema*. And we check whether such a pokémon exists in our database and is displayed on the map (the pokémon must have the active status):

```
catch = function(self, pokemon_id, player)
    -- check player data
    local ok, tuple = self.player_model.flatten(player)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    if not ok then
        return false
    end
    -- get pokemon data
    local p_tuple = box.space.pokemons:get(pokemon_id)
    if p_tuple == nil then
        return false
    end
    local ok, pokemon = self.pokemon_model.unflatten(p_tuple)
    if not ok then
        return false
    end
    if pokemon.status ~= self.state.ACTIVE then
        return false
    end
    -- more catch logic to follow
    <...>
end
```

Next, we calculate the answer: caught or not.

To work with geographical coordinates, we use Tarantool gis module.

To keep things simple, we don't load any specific map, assuming that we deal with a world map. And we do not validate incoming coordinates, assuming again that all received locations are within the planet Earth.

We use two geo-specific variables:

- `wgs84`, which stands for the latest revision of the World Geodetic System standard, WGS84. Basically, it comprises a standard coordinate system for the Earth and represents the Earth as an ellipsoid.
- `nationalmap`, which stands for the US National Atlas Equal Area. This is a projected coordinates system based on WGS84. It gives us a zero base for location projection and allows positioning our players and pokémons in meters.

Both these systems are listed in the EPSG Geodetic Parameter Registry, where each system has a unique number. In our code, we assign these listing numbers to respective variables:

```
wgs84 = 4326,
nationalmap = 2163,
```

For our game logic, we need one more variable, `catch_distance`, which defines how close a player must get to a pokémon before trying to catch it. Let's set the distance to 100 meters.

```
catch_distance = 100,
```

Now we're ready to calculate the answer. We need to project the current location of both player (`p_pos`) and pokémon (`m_pos`) on the map, check whether the player is close enough to the pokémon (using `catch_distance`), and calculate whether the player has caught the pokémon (here we generate some random value and let the pokémon escape if the random value happens to be less than 100 minus pokémon's chance value):

```
-- project locations
local m_pos = gis.Point(
    {pokemon.location.x, pokemon.location.y}, self.wgs84
```

(continues on next page)

(продолжение с предыдущей страницы)

```
):transform(self.nationalmap)
local p_pos = gis.Point(
    {player.location.x, player.location.y}, self.wgs84
):transform(self.nationalmap)

-- check catch distance condition
if p_pos:distance(m_pos) > self.catch_distance then
    return false
end
-- try to catch pokemon
local caught = math.random(100) >= 100 - pokemon.chance
if caught then
    -- update and notify on success
    box.space.pokemons:update(
        pokemon_id, {{'=', self.STATUS, self.state.CAUGHT}}
    )
    self:notify(player, pokemon)
end
return caught
```

### Index iterators

By our gameplay, all caught pokémons are returned back to the map. We do this for all pokémons on the map every 60 seconds using `respawn()` method. We iterate through pokémons by status using Tarantool index iterator function *index_object:pairs()* and reset the statuses of all «caught» pokémons back to «active» using `box.space.pokemons:update()`.

```
respawn = function(self)
    fiber.name('Respawn fiber')
    for _, tuple in box.space.pokemons.index.status:pairs(
           self.state.CAUGHT) do
        box.space.pokemons:update(
            tuple[self.ID],
            {{'=', self.STATUS, self.state.ACTIVE}}
        )
    end
 end
```

For readability, we introduce named fields:

ID = 1, STATUS = 2,

The complete implementation of `start()` now looks like this:

```
-- create game object
start = function(self)
    -- create spaces and indexes
    box.once('init', function()
       box.schema.create_space('pokemons')
       box.space.pokemons:create_index(
           "primary", {type = 'hash', parts = {1, 'unsigned'}}
       )
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        box.space.pokemons:create_index(
            "status", {type = "tree", parts = {2, 'str'}}
        )
    end)

    -- create models
    local ok_m, pokemon = avro.create(schema.pokemon)
    local ok_p, player = avro.create(schema.player)
    if ok_m and ok_p then
        -- compile models
        local ok_cm, compiled_pokemon = avro.compile(pokemon)
        local ok_cp, compiled_player = avro.compile(player)
        if ok_cm and ok_cp then
            -- start the game
            self.pokemon_model = compiled_pokemon
            self.player_model = compiled_player
            self.respawn()
            log.info('Started')
            return true
         else
            log.error('Schema compilation failed')
         end
    else
        log.info('Schema creation failed')
    end
    return false
end
```

### Fibers, yields and cooperative multitasking

But wait! If we launch it as shown above – `self.respawn()` – the function will be executed only once, just like all the other methods. But we need to execute `respawn()` every 60 seconds. Creating a *fiber* is the Tarantool way of making application logic work in the background at all times.

A **fiber** is a set of instructions that are executed with *cooperative multitasking*: the instructions contain *yield* signals, upon which control is passed to another fiber.

Let's launch `respawn()` in a fiber to make it work in the background all the time. To do so, we'll need to amend `respawn()`:

```
respawn = function(self)
    -- let's give our fiber a name;
    -- this will produce neat output in fiber.info()
    fiber.name('Respawn fiber')
    while true do
        for _, tuple in box.space.pokemons.index.status:pairs(
                self.state.CAUGHT) do
            box.space.pokemons:update(
                tuple[self.ID],
                {{'=', self.STATUS, self.state.ACTIVE}}
            )
        end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        fiber.sleep(self.respawn_time)
    end
end
```

and call it as a fiber in `start()`:

```
start = function(self)
    -- create spaces and indexes
        <...>
    -- create models
        <...>
    -- compile models
        <...>
    -- start the game
       self.pokemon_model = compiled_pokemon
       self.player_model = compiled_player
       fiber.create(self.respawn, self)
       log.info('Started')
    -- errors if schema creation or compilation fails
       <...>
end
```

### Logging

One more helpful function that we used in `start()` was `log.info()` from Tarantool *log* module. We also need this function in `notify()` to add a record to the log file on every successful catch:

```
-- event notification
notify = function(self, player, pokemon)
    log.info("Player '%s' caught '%s'", player.name, pokemon.name)
end
```

We use default Tarantool *log settings*, so we'll see the log output in console when we launch our application in script mode.

Great! We've discussed all programming practices used in our Lua module (see pokemon.lua).

Now let's prepare the test environment. As planned, we write a Lua application (see game.lua) to initialize Tarantool's database module, initialize our game, call the game loop and simulate a couple of player requests.

To launch our microservice, we put both the `pokemon.lua` module and the `game.lua` application in the current directory, install all external modules, and launch the Tarantool instance running our `game.lua` application (this example is for Ubuntu):

```
$ ls
game.lua  pokemon.lua
$ sudo apt-get install tarantool-gis
$ sudo apt-get install tarantool-avro-schema
$ tarantool game.lua
```

Tarantool starts and initializes the database. Then Tarantool executes the demo logic from `game.lua`: adds a pokémon named Pikachu (its chance to be caught is very high, 99.1), displays the current map (it contains

one active pokémon, Pikachu) and processes catch requests from two players. Player1 is located just near the lonely Pikachu pokémon and Player2 is located far away from it. As expected, the catch results in this output are «true» for Player1 and «false» for Player2. Finally, Tarantool displays the current map which is empty, because Pikachu is caught and temporarily inactive:

```
$ tarantool game.lua
2017-01-09 20:19:24.605 [6282] main/101/game.lua C> version 1.7.3-43-gf5fa1e1
2017-01-09 20:19:24.605 [6282] main/101/game.lua C> log level 5
2017-01-09 20:19:24.605 [6282] main/101/game.lua I> mapping 1073741824 bytes for tuple␣
↪arena...
2017-01-09 20:19:24.609 [6282] main/101/game.lua I> initializing an empty data directory
2017-01-09 20:19:24.634 [6282] snapshot/101/main I> saving snapshot `./
↪00000000000000000000.snap.inprogress'
2017-01-09 20:19:24.635 [6282] snapshot/101/main I> done
2017-01-09 20:19:24.641 [6282] main/101/game.lua I> ready to accept requests
2017-01-09 20:19:24.786 [6282] main/101/game.lua I> Started
---
- {'id': 1, 'status': 'active', 'location': {'y': 2, 'x': 1}, 'name': 'Pikachu', 'chance
↪': 99.1}
...

2017-01-09 20:19:24.789 [6282] main/101/game.lua I> Player 'Player1' caught 'Pikachu'
true
false
--- []
...

2017-01-09 20:19:24.789 [6282] main C> entering the event loop
```

### nginx

In the real life, this microservice would work over HTTP. Let's add nginx web server to our environment and make a similar demo. But how do we make Tarantool methods callable via REST API? We use nginx with Tarantool nginx upstream module and create one more Lua script (app.lua) that exports three of our game methods – `add_pokemon()`, `map()` and `catch()` – as REST endpoints of the nginx upstream module:

```
local game = require('pokemon')
box.cfg{listen=3301}
game:start()

-- add, map and catch functions exposed to REST API
function add(request, pokemon)
    return {
        result=game:add_pokemon(pokemon)
    }
end

function map(request)
    return {
        map=game:map()
    }
end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
function catch(request, pid, player)
    local id = tonumber(pid)
    if id == nil then
        return {result=false}
    end
    return {
        result=game:catch(id, player)
    }
end
```

An easy way to configure and launch nginx would be to create a Docker container based on a Docker image with nginx and the upstream module already installed (see http/Dockerfile). We take a standard nginx.conf, where we define an upstream with our Tarantool backend running (this is another Docker container, see details below):

```
upstream tnt {
      server pserver:3301 max_fails=1 fail_timeout=60s;
      keepalive 250000;
}
```

and add some Tarantool-specific parameters (see descriptions in the upstream module's README file):

```
server {
  server_name tnt_test;

  listen 80 default deferred reuseport so_keepalive=on backlog=65535;

  location = / {
      root /usr/local/nginx/html;
  }

  location /api {
    # answers check infinity timeout
    tnt_read_timeout 60m;
    if ( $request_method = GET ) {
       tnt_method "map";
    }
    tnt_http_rest_methods get;
    tnt_http_methods all;
    tnt_multireturn_skip_count 2;
    tnt_pure_result on;
    tnt_pass_http_request on parse_args;
    tnt_pass tnt;
  }
}
```

Likewise, we put Tarantool server and all our game logic in a second Docker container based on the official Tarantool 1.9 image (see src/Dockerfile) and set the container's default command to `tarantool app.lua`. This is the backend.

### Non-blocking IO

To test the REST API, we create a new script (client.lua), which is similar to our `game.lua` application, but makes HTTP POST and GET requests rather than calling Lua functions:

```
local http = require('curl').http()
local json = require('json')
local URI = os.getenv('SERVER_URI')
local fiber = require('fiber')

local player1 = {
    name="Player1",
    id=1,
    location = {
        x=1.0001,
        y=2.0003
    }
}
local player2 = {
    name="Player2",
    id=2,
    location = {
        x=30.123,
        y=40.456
    }
}

local pokemon = {
    name="Pikachu",
    chance=99.1,
    id=1,
    status="active",
    location = {
        x=1,
        y=2
    }
}

function request(method, body, id)
    local resp = http:request(
        method, URI, body
    )
    if id ~= nil then
        print(string.format('Player %d result: %s',
            id, resp.body))
    else
        print(resp.body)
    end
end

local players = {}
function catch(player)
    fiber.sleep(math.random(5))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    print('Catch pokemon by player ' .. tostring(player.id))
    request(
        'POST', '{"method": "catch",
        "params": [1, '..json.encode(player)..']}',
        tostring(player.id)
    )
    table.insert(players, player.id)
end

print('Create pokemon')
request('POST', '{"method": "add",
    "params": ['..json.encode(pokemon)..']}')
request('GET', '')

fiber.create(catch, player1)
fiber.create(catch, player2)

-- wait for players
while #players ~= 2 do
    fiber.sleep(0.001)
end

request('GET', '')
os.exit()
```

When you run this script, you'll notice that both players have equal chances to make the first attempt at catching the pokémon. In a classical Lua script, a networked call blocks the script until it's finished, so the first catch attempt can only be done by the player who entered the game first. In Tarantool, both players play concurrently, since all modules are integrated with Tarantool *cooperative multitasking* and use non-blocking I/O.

Indeed, when Player1 makes its first REST call, the script doesn't block. The fiber running `catch()` function on behalf of Player1 issues a non-blocking call to the operating system and yields control to the next fiber, which happens to be the fiber of Player2. Player2's fiber does the same. When the network response is received, Player1's fiber is activated by Tarantool cooperative scheduler, and resumes its work. All Tarantool *modules* use non-blocking I/O and are integrated with Tarantool cooperative scheduler. For module developers, Tarantool provides an *API*.

For our HTTP test, we create a third container based on the official Tarantool 1.9 image (see client/Dockerfile) and set the container's default command to `tarantool client.lua`.

To run this test locally, download our pokemon project from GitHub and say:

```
$ docker-compose build
$ docker-compose up
```

Docker Compose builds and runs all the three containers: `pserver` (Tarantool backend), `phttp` (nginx) and `pclient` (demo client). You can see log messages from all these containers in the console, pclient saying that it made an HTTP request to create a pokémon, made two catch requests, requested the map (empty since the pokémon is caught and temporarily inactive) and exited:

```
pclient_1   | Create pokemon
<...>
pclient_1   | {"result":true}
pclient_1   | {"map":[{"id":1,"status":"active","location":{"y":2,"x":1},"name":"Pikachu",
↪"chance":99.100000}]}
pclient_1   | Catch pokemon by player 2
pclient_1   | Catch pokemon by player 1
pclient_1   | Player 1 result: {"result":true}
pclient_1   | Player 2 result: {"result":false}
pclient_1   | {"map":[]}
pokemon_pclient_1 exited with code 0
```

Congratulations! Here's the end point of our walk-through. As further reading, see more about *installing* and *contributing* a module.

See also reference on *Tarantool modules* and *C API*, and don't miss our *Lua cookbook recipes*.

### 3.8.3 Разработка с IDE

Для разработки и отладки Lua-приложений для Tarantool можно использовать IntelliJ IDEA в качестве интегрированной среды разработки (IDE).

1. Загрузите и установите IDE с официального сайта.

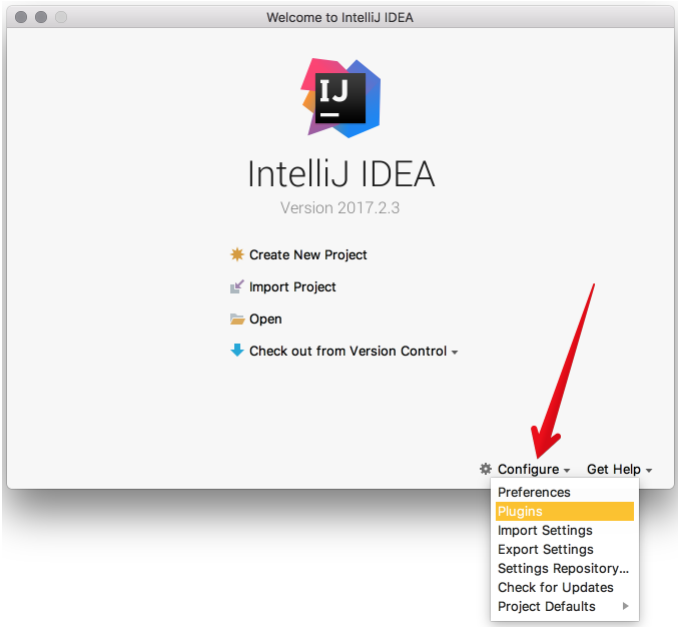
   JetBrains предоставляет специализированные версии для разных языков программирования: IntelliJ IDEA (Java), PHPStorm (PHP), PyCharm (Python), RubyMine (Ruby), CLion (C/C++), WebStorm (Web) и другие. Поэтому загрузите версию, которая подходит предпочитаемому языку.

   Для всех версий поддерживается интеграция с Tarantool.

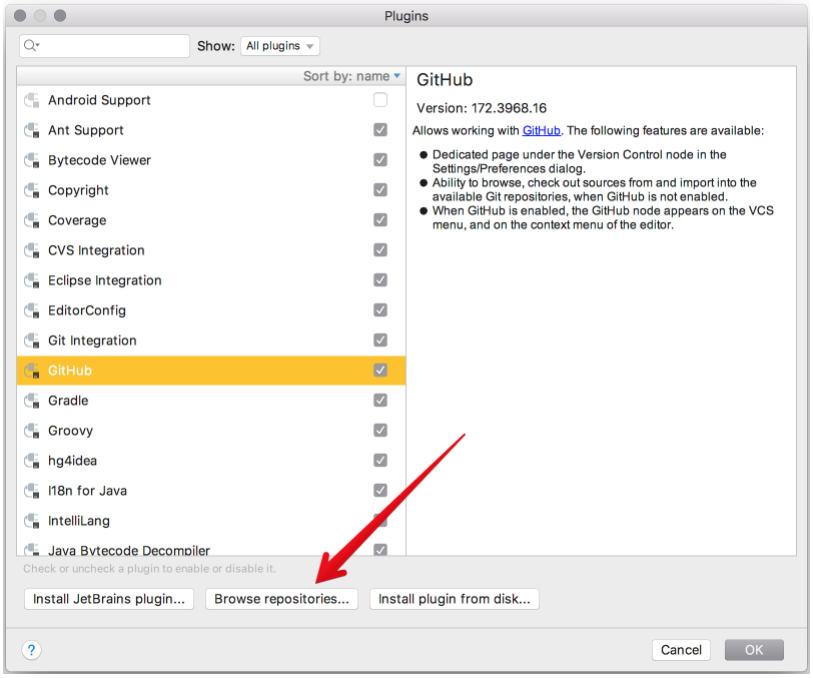
2. Настройте IDE:

   a. Запустите IntelliJ IDEA.

   b. Нажмите кнопку `Configure` и выберите `Plugins`.



c. Нажмите `Browse repositories`.

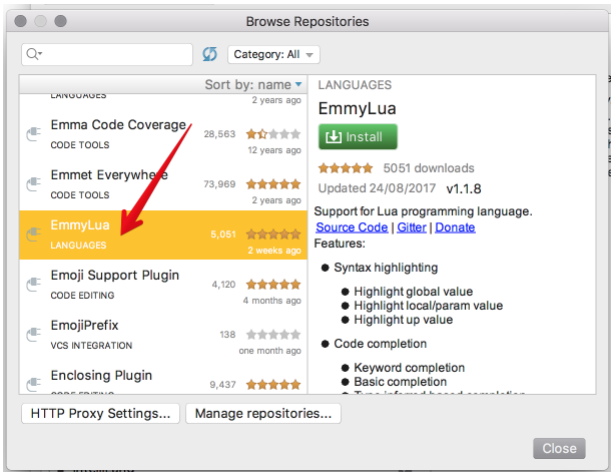



d. Установите плагин `EmmyLua`.

---

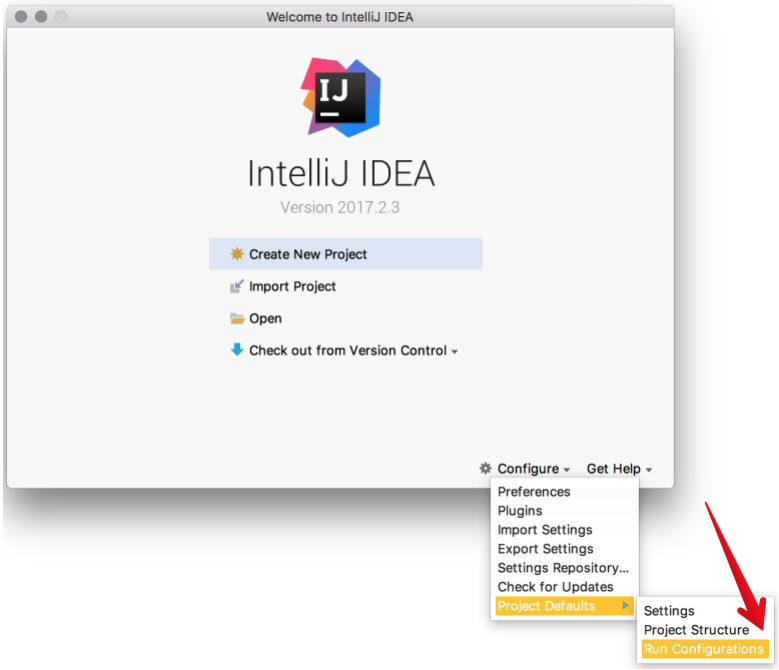
**Примечание:** Не путайте с плагином `Lua`, у которого меньше возможностей, чем у `EmmyLua`.

---




e. Перезапустите IntelliJ IDEA.

f. Нажмите `Configure`, выберите `Project Defaults`, а затем `Run Configurations`.

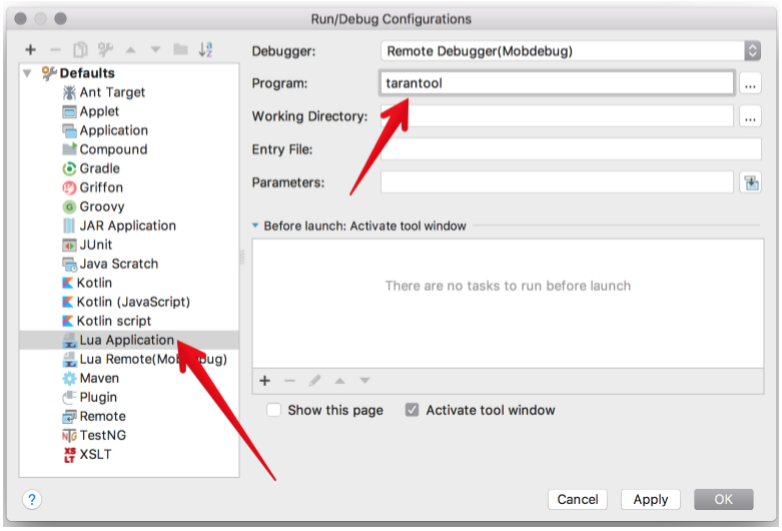



g. Найдите `Lua Application` в боковой панели слева.

h. В `Program` введите путь к установленному бинарному файлу `tarantool`.

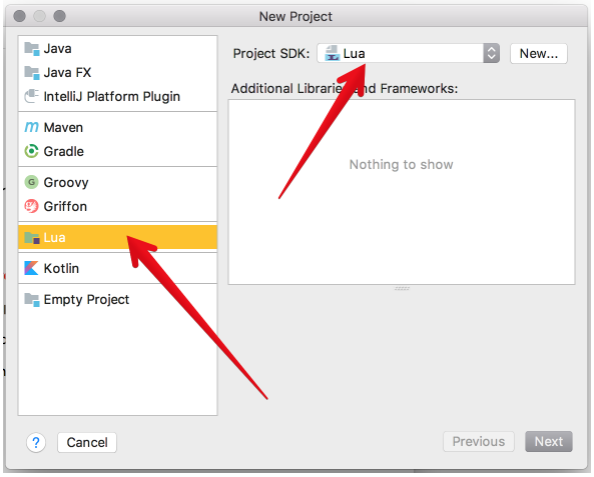
   По умолчанию, это `tarantool` или `/usr/bin/tarantool` на большинстве платформ.

   Если вы установили `tarantool` из источников в другую директорию, укажите здесь правильный путь.

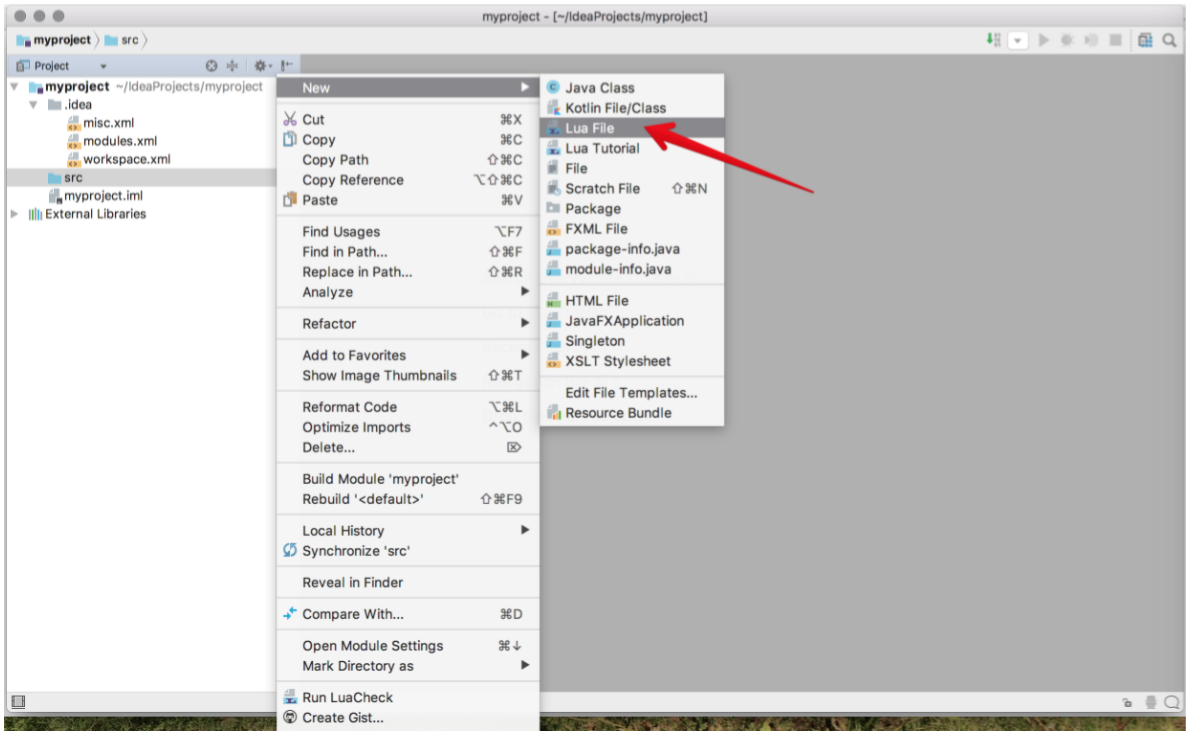



Теперь IntelliJ IDEA можно использовать с Tarantool.

3. Создайте новый проект на Lua.

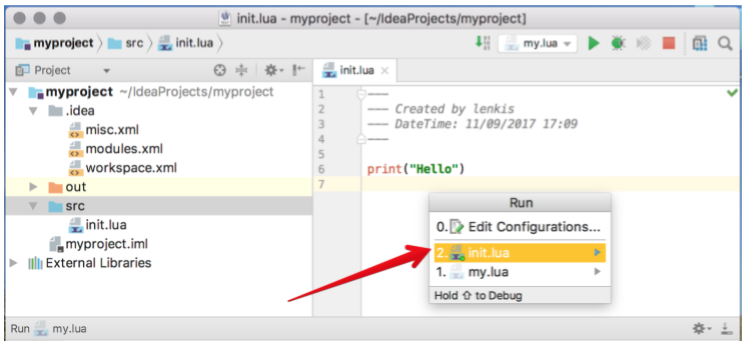



4. Добавьте новый Lua-файл, например, `init.lua`.




5. Разработайте код, сохраните файл.

6. Чтобы запустить приложение, нажмите `Run -> Run` в основном меню и выберите исходный файл из списка.

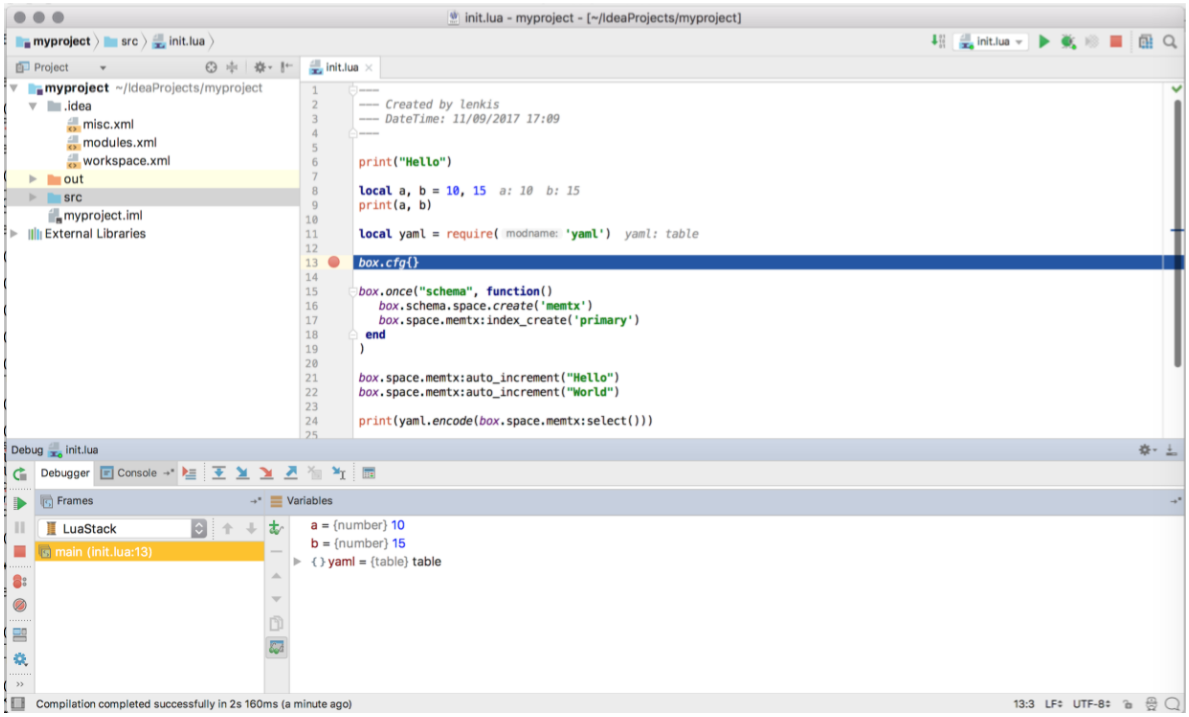



Или нажмите `Run -> Debug` для начала отладки.

---

**Примечание:** Чтобы использовать Lua-отладчик, обновите Tarantool до версии 1.7.5-29-gbb6170e4b или более поздней версии.

---




### 3.8.4 Примеры и рекомендации по разработке на Lua

Ниже представлены дополнения в виде Lua-программ для часто встречающихся или сложных случаев.

Любую из этих программ можно выполнить, скопировав код в `.lua-файл`, а затем выполнив в командной строке `chmod +x ./имя-программы.lua` и :samp :`./{имя-программы}.lua`.

Первая строка – это шебанг:

```
#!/usr/bin/env tarantool
```

Он запускает сервер приложений Tarantool на языке Lua, который должен быть в пути выполнения.

В этом разделе собраны следующие рецепты:

- *hello_world.lua*
- *console_start.lua*
- *fio_read.lua*
- *fio_write.lua*
- *ffi_printf.lua*
- *ffi_gettimeofday.lua*
- *ffi_zlib.lua*
- *ffi_meta.lua*
- *ffi_varbinary_insert.lua*
- *print_arrays.lua*
- *count_array.lua*
- *count_array_with_nils.lua*
- *count_array_with_nulls.lua*
- *count_map.lua*
- *swap.lua*
- *class.lua*
- *garbage.lua*
- *fiber_producer_and_consumer.lua*
- *socket_tcpconnect.lua*
- *socket_tcp_echo.lua*
- *getaddrinfo.lua*
- *socket_udp_echo.lua*
- *http_get.lua*
- *http_send.lua*
- *http_server.lua*
- *http_generate_html.lua*
- *select_all.go*

Можно использовать свободно.

Другие рецепты см. на GitHub Tarantool.

### hello_world.lua

Стандартный пример простой программы.

```
#!/usr/bin/env tarantool

 print('Hello, World!')
```

### console_start.lua

Для инициализации базы данных (создания спейсов) используйте *box.once()*, если сервер запускается впервые. Затем используйте *console.start()*, чтобы запустить интерактивный режим.

```
#!/usr/bin/env tarantool

-- Настроить базу данных
box.cfg {
    listen = 3313
}

box.once("bootstrap", function()
    box.schema.space.create('tweedledum')
    box.space.tweedledum:create_index('primary',
        { type = 'TREE', parts = {1, 'unsigned'}})
end)

require('console').start()
```

### fio_read.lua

Используйте *Модуль fio*, чтобы открыть, прочитать и закрыть файл.

```
#!/usr/bin/env tarantool

local fio = require('fio')
local errno = require('errno')
local f = fio.open('/tmp/xxxx.txt', {'O_RDONLY' })
if not f then
    error("Failed to open file: "..errno.strerror())
end
local data = f:read(4096)
f:close()
print(data)
```

### fio_write.lua

Используйте *Модуль fio*, чтобы открыть, записать данные и закрыть файл.

```
#!/usr/bin/env tarantool

local fio = require('fio')
local errno = require('errno')
local f = fio.open('/tmp/xxxx.txt', {'O_CREAT', 'O_WRONLY', 'O_APPEND'},
    tonumber('0666', 8))
if not f then
    error("Failed to open file: "..errno.strerror())
end
f:write("Hello\n");
f:close()
```

### ffi_printf.lua

Используйте Библиотеку LuaJIT FFI, чтобы вызвать встроенную в C функцию: printf(). (Чтобы лучше понимать FFI, см. Учебное пособие по FFI.)

```
#!/usr/bin/env tarantool

local ffi = require('ffi')
ffi.cdef[[
    int printf(const char *format, ...);
]]

ffi.C.printf("Hello, %s\n", os.getenv("USER"));
```

### ffi_gettimeofday.lua

Используйте Библиотеку LuaJIT FFI, чтобы вызвать встроенную в C функцию: gettimeofday(). Она позволяет получить значение времени с точностью в миллисекундах, в отличие от функции времени в Tarantool *Модуль clock*.

```
#!/usr/bin/env tarantool

local ffi = require('ffi')
ffi.cdef[[
    typedef long time_t;
    typedef struct timeval {
    time_t tv_sec;
    time_t tv_usec;
} timeval;
    int gettimeofday(struct timeval *t, void *tzp);
]]

local timeval_buf = ffi.new("timeval")
local now = function()
    ffi.C.gettimeofday(timeval_buf, nil)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    return tonumber(timeval_buf.tv_sec * 1000 + (timeval_buf.tv_usec / 1000))
end
```

### ffi_zlib.lua

Используйте Библиотеку LuaJIT FFI, чтобы вызвать библиотечную функцию в C. (Чтобы лучше понимать FFI, см. Учебное пособие по FFI.)

```
#!/usr/bin/env tarantool

local ffi = require("ffi")
ffi.cdef[[
    unsigned long compressBound(unsigned long sourceLen);
    int compress2(uint8_t *dest, unsigned long *destLen,
    const uint8_t *source, unsigned long sourceLen, int level);
    int uncompress(uint8_t *dest, unsigned long *destLen,
    const uint8_t *source, unsigned long sourceLen);
]]
local zlib = ffi.load(ffi.os == "Windows" and "zlib1" or "z")

-- Надстройка Lua для функции compress2()
local function compress(txt)
    local n = zlib.compressBound(#txt)
    local buf = ffi.new("uint8_t[?]", n)
    local buflen = ffi.new("unsigned long[1]", n)
    local res = zlib.compress2(buf, buflen, txt, #txt, 9)
    assert(res == 0)
    return ffi.string(buf, buflen[0])
end

-- Надстройка Lua для функции uncompress
local function uncompress(comp, n)
    local buf = ffi.new("uint8_t[?]", n)
    local buflen = ffi.new("unsigned long[1]", n)
    local res = zlib.uncompress(buf, buflen, comp, #comp)
    assert(res == 0)
    return ffi.string(buf, buflen[0])
end

-- Простой код теста
local txt = string.rep("abcd", 1000)
print("Uncompressed size: ", #txt)
local c = compress(txt)
print("Compressed size: ", #c)
local txt2 = uncompress(c, #txt)
assert(txt2 == txt)
```

### ffi_meta.lua

Используйте Библиотеку LuaJIT FFI, чтобы получить доступ к объекту в C с помощью метаметода (метод, который определен метатаблицей).

```
#!/usr/bin/env tarantool

local ffi = require("ffi")
ffi.cdef[[
typedef struct { double x, y; } point_t;
]]

local point
local mt = {
  __add = function(a, b) return point(a.x+b.x, a.y+b.y) end,
  __len = function(a) return math.sqrt(a.x*a.x + a.y*a.y) end,
  __index = {
    area = function(a) return a.x*a.x + a.y*a.y end,
  },
}
point = ffi.metatype("point_t", mt)

local a = point(3, 4)
print(a.x, a.y)  --> 3  4
print(#a)        --> 5
print(a:area())  --> 25
local b = a + point(0.5, 8)
print(#b)        --> 12.5
```

### ffi_varbinary_insert.lua

Используйте библиотеку LuaJIT ffi для вставки кортежа, имеющего поле VARBINARY.

Обратите внимание, что это разрешено только внутри memtx-транзакции: когда `box_insert()` не передает управление.

Lua не имеет прямой поддержки VARBINARY, поэтому использование C является одним из способов вставить данные, которые в MessagePack хранятся в виде bin (MP_BIN). Если кортеж будет получен позже, то поле «b» будет иметь тип = «cdata».

```
#!/usr/bin/env tarantool

-- здесь должен быть box.cfg{}

s = box.schema.space.create('withdata')
s:format({{"b", "varbinary"}})
s:create_index('pk', {parts = {1, "varbinary"}})

buffer = require('buffer')
ffi = require('ffi')

function varbinary_insert(space, bytes)
    local tmpbuf = buffer.ibuf()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    local p = tmpbuf:alloc(3 + #bytes)
    p[0] = 0x91 -- MsgPack code for "array-1"
    p[1] = 0xC4 -- MsgPack code for "bin-8" so up to 256 bytes
    p[2] = #bytes
    for i, c in pairs(bytes) do p[i + 3 - 1] = c end
    ffi.cdef[[int box_insert(uint32_t space_id,
                             const char *tuple,
                             const char *tuple_end,
                             box_tuple_t **result);]]
    ffi.C.box_insert(space.id, tmpbuf.rpos, tmpbuf.wpos, nil)
    tmpbuf:recycle()
end

varbinary_insert(s, {0xDE, 0xAD, 0xBE, 0xAF})
varbinary_insert(s, {0xFE, 0xED, 0xFA, 0xCE})

-- если сработает, Tarantool войдет в цикл событий
```

### print_arrays.lua

Используйте, чтобы создать Lua-таблицы и вывести их. Следует отметить, что для таблицы типа массива (array) функция-итератор будет `ipairs()`, а для таблицы типа ассоциативного массива (map) функция-итератор – `pairs()`. (`ipairs()` быстрее, чем `pairs()`, но `pairs()` рекомендуется для ассоциативных массивов или смешанных таблиц.) Результат будет выглядеть следующим образом: «1 Apple | 2 Orange | 3 Grapefruit | 4 Banana | k3 v3 | k1 v1 | k2 v2».

```
#!/usr/bin/env tarantool

array = { 'Apple', 'Orange', 'Grapefruit', 'Banana'}
for k, v in ipairs(array) do print(k, v) end

map = { k1 = 'v1', k2 = 'v2', k3 = 'v3' }
for k, v in pairs(map) do print(k, v) end
```

### count_array.lua

Используйте оператор „#“, чтобы получить количество элементов в Lua-таблице типа массива. У этой операции сложность O(log(N)).

```
#!/usr/bin/env tarantool

array = { 1, 2, 3}
print(#array)
```

#### count_array_with_nils.lua

Отсутствующие элементы в массивах, которые Lua рассматривает как nil, заставляют простой оператор „#“ выдавать неправильные результаты. Команда «print(#t)» выведет «4», команда «print(counter)» выведет «3», а команда «print(max)» – «10». Другие табличные функции, такие как *table.sort()*, также сработают неправильно при наличии нулевых значений nil.

```
#!/usr/bin/env tarantool

local t = {}
t[1] = 1
t[4] = 4
t[10] = 10
print(#t)
local counter = 0
for k,v in pairs(t) do counter = counter + 1 end
print(counter)
local max = 0
for k,v in pairs(t) do if k > max then max = k end end
print(max)
```

#### count_array_with_nulls.lua

Используйте явные значения``NULL``, чтобы избежать проблем, вызванных nil в Lua == поведение с пропущенными значениями. Хотя `json.NULL == nil` является `true`, все команды вывода в данной программе выведут правильное значение: 10.

```
#!/usr/bin/env tarantool

local json = require('json')
local t = {}
t[1] = 1; t[2] = json.NULL; t[3]= json.NULL;
t[4] = 4; t[5] = json.NULL; t[6]= json.NULL;
t[6] = 4; t[7] = json.NULL; t[8]= json.NULL;
t[9] = json.NULL
t[10] = 10
print(#t)
local counter = 0
for k,v in pairs(t) do counter = counter + 1 end
print(counter)
local max = 0
for k,v in pairs(t) do if k > max then max = k end end
print(max)
```

### count_map.lua

Программа используется для получения количества элементов в таблице типа ассоциативного массива.

```
#!/usr/bin/env tarantool

local map = { a = 10, b = 15, c = 20 }
local size = 0
for _ in pairs(map) do size = size + 1; end
print(size)
```

### swap.lua

Программа использует особенность Lua менять местами две переменные без необходимости использования третьей переменной.

```
#!/usr/bin/env tarantool

local x = 1
local y = 2
x, y = y, x
print(x, y)
```

### class.lua

Используется для создания класса, метатаблицы для класса, экземпляра класса. Другой пример можно найти в http://lua-users.org/wiki/LuaClassesWithMetatable.

```
#!/usr/bin/env tarantool

-- определить объекты класса
local myclass_somemethod = function(self)
    print('test 1', self.data)
end

local myclass_someothermethod = function(self)
    print('test 2', self.data)
end

local myclass_tostring = function(self)
    return 'MyClass <'..self.data..'>'
end

local myclass_mt = {
    __tostring = myclass_tostring;
    __index = {
        somemethod = myclass_somemethod;
        someothermethod = myclass_someothermethod;
    }
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- создать новый объект своего класса myclass
local object = setmetatable({ data = 'data'}, myclass_mt)
print(object:somemethod())
print(object.data)
```

### garbage.lua

Запустите сборщик мусора в Lua с помощью функции collectgarbage.

```
#!/usr/bin/env tarantool

collectgarbage('collect')
```

### fiber_producer_and_consumer.lua

Запустите один файбер для производителя и один файбер для потребителя. Используйте *fiber.channel()* для обмена данных и синхронизации. Можно настроить ширину канала (`ch_size` в программном коде) для управления количеством одновременных задач к обработке.

```
#!/usr/bin/env tarantool

local fiber = require('fiber')
local function consumer_loop(ch, i)
    -- инициализировать потребитель синхронно или выдать ошибку()
    fiber.sleep(0) -- позволить fiber.create() продолжать
    while true do
        local data = ch:get()
        if data == nil then
            break
        end
        print('consumed', i, data)
        fiber.sleep(math.random()) -- моделировать работу
    end
end

local function producer_loop(ch, i)
    -- инициализировать потребитель синхронно или выдать ошибку()
    fiber.sleep(0) -- allow fiber.create() to continue
    while true do
        local data = math.random()
        ch:put(data)
        print('produced', i, data)
    end
end

local function start()
    local consumer_n = 5
    local producer_n = 3

    -- создать канал
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    local ch_size = math.max(consumer_n, producer_n)
    local ch = fiber.channel(ch_size)

    -- запустить потребители
    for i=1, consumer_n,1 do
        fiber.create(consumer_loop, ch, i)
    end

    -- запустить производители
    for i=1, producer_n,1 do
        fiber.create(producer_loop, ch, i)
    end
end

start()
print('started')
```

### socket_tcpconnect.lua

Используйте *socket.tcp_connect()* для подключения к удаленному серверу по TCP. Можно отобразить информацию о подключении и результат запроса GET.

```
#!/usr/bin/env tarantool

local s = require('socket').tcp_connect('google.com', 80)
print(s:peer().host)
print(s:peer().family)
print(s:peer().type)
print(s:peer().protocol)
print(s:peer().port)
print(s:write("GET / HTTP/1.0\r\n\r\n"))
print(s:read('\r\n'))
print(s:read('\r\n'))
```

### socket_tcp_echo.lua

Используйте *socket.tcp_connect()* для настройки простого TCP-сервера путем создания функции, которая обрабатывает запросы и отражает их, а затем передачи функции на *socket.tcp_server()*. Данная программа была протестирована на 100 000 клиентов, каждый из которых получил отдельный файбер.

```
#!/usr/bin/env tarantool

local function handler(s, peer)
    s:write("Welcome to test server, " .. peer.host .."\n")
    while true do
        local line = s:read('\n')
        if line == nil then
            break -- ошибка или конец файла
        end
        if not s:write("pong: "..line) then
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            break -- ошибка или конец файла
        end
    end
end

local server, addr = require('socket').tcp_server('localhost', 3311, handler)
```

### getaddrinfo.lua

Используйте *socket.getaddrinfo()*, чтобы провести неблокирующее разрешение имен DNS, получая как AF_INET6, так и AF_INET информацию для „google.com“. Данная техника не всегда необходима для TCP-соединений, поскольку *socket.tcp_connect()* выполняет `socket.getaddrinfo` с точки зрения внутреннего устройства до попытки соединения с первым доступным адресом.

```
#!/usr/bin/env tarantool

local s = require('socket').getaddrinfo('google.com', 'http', { type = 'SOCK_STREAM' })
print('host=',s[1].host)
print('family=',s[1].family)
print('type=',s[1].type)
print('protocol=',s[1].protocol)
print('port=',s[1].port)
print('host=',s[2].host)
print('family=',s[2].family)
print('type=',s[2].type)
print('protocol=',s[2].protocol)
print('port=',s[2].port)
```

### socket_udp_echo.lua

В данный момент в Tarantool нет функции `udp_server`, поэтому socket_udp_echo.lua – более сложная программа, чем socket_tcp_echo.lua. Ее можно реализовать с помощью сокетов и файберов.

```
#!/usr/bin/env tarantool

local socket = require('socket')
local errno = require('errno')
local fiber = require('fiber')

local function udp_server_loop(s, handler)
    fiber.name("udp_server")
    while true do
        -- попытка прочитать сначала датаграмму
        local msg, peer = s:recvfrom()
        if msg == "" then
            -- сокет был закрыт с помощью s:close()
            break
        elseif msg ~= nil then
            -- получена новая датаграмма
            handler(s, peer, msg)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        else
            if s:errno() == errno.EAGAIN or s:errno() == errno.EINTR then
                -- сокет не готов
                s:readable() -- передача управления, epoll сообщит, когда будут новые␣
↪данные
            else
                -- ошибка сокета
                local msg = s:error()
                s:close() -- сохранить ресурсы и не ждать сборку мусора
                error("Socket error: " .. msg)
            end
        end
    end
end

local function udp_server(host, port, handler)
    local s = socket('AF_INET', 'SOCK_DGRAM', 0)
    if not s then
        return nil -- проверить номер ошибки errno:strerror()
   end
    if not s:bind(host, port) then
        local e = s:errno() -- сохранить номер ошибки errno
        s:close()
        errno(e) -- восстановить номер ошибки errno
        return nil -- проверить номер ошибки errno:strerror()
    end

    fiber.create(udp_server_loop, s, handler) -- запустить новый файбер в фоновом режиме
    return s
end
```

Функция для клиента, который подключается к этому серверу, может выглядеть следующим образом:

```
local function handler(s, peer, msg)
    -- Необязательно ждать, пока сокет будет готов отправлять UDP
    -- s:writable()
    s:sendto(peer.host, peer.port, "Pong: " .. msg)
end

local server = udp_server('127.0.0.1', 3548, handler)
if not server then
    error('Failed to bind: ' .. errno.strerror())
end

print('Started')

require('console').start()
```

#### http_get.lua

Используйте *Модуль HTTP* для получения данных по HTTP.

```
#!/usr/bin/env tarantool

local http_client = require('http.client')
local json = require('json')
local r = http_client.get('https://api.frankfurter.app/latest?to=USD%2CRUB')
if r.status ~= 200 then
    print('Failed to get currency ', r.reason)
    return
end
local data = json.decode(r.body)
print(data.base, 'rate of', data.date, 'is', data.rates.RUB, 'RUB or', data.rates.USD,
↪'USD')
```

#### http_send.lua

Используйте *Модуль HTTP* для отправки данных по HTTP.

```
#!/usr/bin/env tarantool

local http_client = require('http.client')
local json = require('json')
local data = json.encode({ Key = 'Value'})
local headers = { Token = 'xxxx', ['X-Secret-Value'] = '42' }
local r = http_client.post('http://localhost:8081', data, { headers = headers})
if r.status == 200 then
    print 'Success'
end
```

#### http_server.lua

Используйте сторонний модуль http (который необходимо предварительно установить), чтобы превратить Tarantool в веб-сервер.

```
#!/usr/bin/env tarantool

local function handler(self)
    return self:render{ json = { ['Your-IP-Is'] = self.peer.host } }
end

local server = require('http.server').new(nil, 8080, {charset = "utf8"}) --␣
↪прослушивание *:8080
server:route({ path = '/' }, handler)
server:start()
-- подключение к localhost:8080 и просмотр json
```

### http_generate_html.lua

Use the http `rock` (which must first be installed) to generate HTML pages from templates. The http rock has a fairly simple template engine which allows execution of regular Lua code inside text blocks (like PHP). Therefore there is no need to learn new languages in order to write templates.

```
#!/usr/bin/env tarantool

local function handler(self)
local fruits = { 'Apple', 'Orange', 'Grapefruit', 'Banana'}
    return self:render{ fruits = fruits }
end

local server = require('http.server').new(nil, 8080, {charset = "utf8"}) -- nil означает
↪'*'
server:route({ path = '/', file = 'index.html.lua' }, handler)
server:start()
```

HTML-файл для этого сервера, включая Lua, может выглядеть следующим образом (будет выведено «1 Apple | 2 Orange | 3 Grapefruit | 4 Banana»). Создайте директорию `templates` и поместите в неё файл:

```
<html>
<body>
    <table border="1">
        % for i,v in pairs(fruits) do
        <tr>
            <td><%= i %></td>
            <td><%= v %></td>
        </tr>
        % end
    </table>
</body>
</html>
```

### select_all.go

На языке Go выборка содержимого всего спейса не является тривиальной задачей, которая решается в одну строчку. Ниже мы приводим пример программы, которая осуществляет полную выборку из спейса „tester“. Эту программу нужно вызвать на том экземпляре, с которым вы собираетесь установить соединение через Go-коннектор.

```
package main

import (
        "fmt"
        "log"

        "github.com/tarantool/go-tarantool"
)

/*
```

(continues on next page)

(продолжение с предыдущей страницы)

```
box.cfg{listen = 3301}
box.schema.user.passwd('pass')

s = box.schema.space.create('tester')
s:format({
    {name = 'id', type = 'unsigned'},
    {name = 'band_name', type = 'string'},
    {name = 'year', type = 'unsigned'}
})
s:create_index('primary', { type = 'hash', parts = {'id'} })
s:create_index('scanner', { type = 'tree', parts = {'id', 'band_name'} })

s:insert{1, 'Roxette', 1986}
s:insert{2, 'Scorpions', 2015}
s:insert{3, 'Ace of Base', 1993}
*/

func main() {
        conn, err := tarantool.Connect("127.0.0.1:3301", tarantool.Opts{
                User: "admin",
                Pass: "pass",
        })

        if err != nil {
                log.Fatalf("Connection refused")
        }
        defer conn.Close()

        spaceName := "tester"
        indexName := "scanner"
        idFn := conn.Schema.Spaces[spaceName].Fields["id"].Id
        bandNameFn := conn.Schema.Spaces[spaceName].Fields["band_name"].Id

        var tuplesPerRequest uint32 = 2
        cursor := []interface{}{}

        for {
                resp, err := conn.Select(spaceName, indexName, 0, tuplesPerRequest,␣
↪tarantool.IterGt, cursor)
                if err != nil {
                        log.Fatalf("Failed to select: %s", err)
                }

                if resp.Code != tarantool.OkCode {
                        log.Fatalf("Select failed: %s", resp.Error)
                }

                if len(resp.Data) == 0 {
                        break
                }

                fmt.Println("Iteration")
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                tuples := resp.Tuples()
                for _, tuple := range tuples {
                        fmt.Printf("\t%v\n", tuple)
                }

                lastTuple := tuples[len(tuples)-1]
                cursor = []interface{}{lastTuple[idFn], lastTuple[bandNameFn]}
        }
}
```

### 3.8.5 Практические задания на Lua

#### Первые шаги

Если вы только осваиваете Lua, рекомендуем выполнить практическое задание, встроенное в Tarantool. Чтобы начать работу с этим заданием, выполните команду `tutorial()` в консоли Tarantool:

```
tarantool> tutorial()
---
- |
 Tutorial -- Screen #1 -- Hello, Moon
 ====================================

 Welcome to the Tarantool tutorial.
 It will introduce you to Tarantool's Lua application server
 and database server, which is what's running what you're seeing.
 This is INTERACTIVE -- you're expected to enter requests
 based on the suggestions or examples in the screen's text.
 <...>
```

#### Вставка 1 млн кортежей с помощью хранимой процедуры на языке Lua

Задание по данному практикуму: "Вставьте 1 миллион кортежей. В каждом кортеже должно быть поле, которое соответствует ключу в первичном индексе, в виде постоянно возрастающего числа, а также поле в виде буквенной строки со случайным значением из 10 символов."

Цель данного упражнения состоит в том, чтобы показать, как выглядят Lua-функции в Tarantool. Необходимо будет работать с математической библиотекой Lua, библиотекой для работы со строками интерпретатора Lua, Tarantool-библиотекой `box`, Tarantool-библиотекой `box.tuple`, циклами и конкатенацией. Инструкции легко будет выполнять даже тем, кто никогда не использовал раньше Lua или Tarantool. Единственное требование – знание того, как работают другие языки программирования, и изучение первых двух глав данного руководства. Но для лучшего понимания можно следовать по комментариям и ссылкам на руководство по Lua или другим пунктам в данном руководстве по Tarantool. А чтобы облегчить изучение, читайте инструкции параллельно с вводом операторов в Tarantool-клиент.

### Настройка

Будем использовать Tarantool-песочницу, которую создавали для *упражнений раздела «Руководство для начинающих»*. Таким образом, у нас есть один спейс и числовой ключ первичного индекса, а также экземпляр Tarantool, который также выступает в виде клиента.

### Разделитель

В более ранних версиях Tarantool многострочные функции обрамляются символами-разделителями. Сейчас в них нет необходимости, поэтому в данном практическом задании они использоваться не будут. Однако они все еще поддерживаются. Если вы хотите использовать разделители или используете более раннюю версию Tarantool, перед работой проверьте описание синтаксиса для *объявления разделителя*.

### Создание функции, которая возвращает строку

Начнем с создания функции, которая возвращает заданную строку – “Hello world”.

```
function string_function()
  return "hello world"
end
```

Слово «function» (функция) – ключевое слово в языке Lua. Рассмотрим подробно работу с языком Lua. Имя функции – string_function (строковая_функция). В функции есть один исполняемый оператор, return "hello world" (вернуть «hello world»). Строка «hello world» здесь заключена в двойные кавычки, хотя в Lua это не имеет значения, можно использовать одинарные кавычки. Слово «end» означает, что “это конец объявления Lua-функции.” Чтобы проверить работу функции, можем выполнить команду

```
string_function()
```

Отправка function-name() (имя-функции) означает команду вызова Lua-функции. В результате возвращаемая функцией строка появится на экране.

Для получения подробной информации о строках в языке Lua, см. Главу 2.4 «Строки» в руководстве по языку Lua. Для получения подробной информации о функциях см. Главу 5 «Функции» в руководстве по языку Lua (chapter 5 «Functions»).

Теперь вывод на экране выглядит следующим образом:

```
tarantool> function string_function()
         >   return "hello world"
         > end
---
...
tarantool> string_function()
---
- hello world
...
tarantool>
```

### Создание функции, которая вызывает другую функцию и определяет переменную

Теперь у нас есть функция `string_function`, и можно вызвать ее с помощью другой функции.

```
function main_function()
  local string_value
  string_value = string_function()
  return string_value
end
```

Сначала объявим переменную «`string_value`» (значение_строки). Слово «`local`» (локально) означает, что string_value появится только в `main_function` (основная_функция). Если бы мы не использовали «`local`», то `string_value` увидели бы даже пользователи других клиентов, которые подключились к данному экземпляру! Иногда это может быть очень полезно при взаимодействии клиентов, но не в нашем случае.

Затем определим значение для `string_value`, а именно, результат функции `string_function()`. Сейчас вызовем `main_function()`, чтобы проверить, что значение определено.

Для получения подробной информации о переменных в языке Lua, см. Главу 4.2 «Локальные переменные и блоки» в руководстве по языку Lua (chapter 4.2 «Local Variables and Blocks»).

Теперь вывод на экране выглядит следующим образом:

```
tarantool> function main_function()
         >   local string_value
         >   string_value = string_function()
         >   return string_value
         > end
---
...
tarantool> main_function()
---
- hello world
...
tarantool>
```

### Изменение функции для возврата строки из одной случайной буквы

Сейчас стало понятно, как задавать переменную, поэтому можно изменить функцию `string_function()` так, чтобы вместо возврата заданной фразы «Hello world», она возвращала случайным образом выбранную букву от „A“ до „Z“.

```
function string_function()
  local random_number
  local random_string
  random_number = math.random(65, 90)
  random_string = string.char(random_number)
  return random_string
end
```

Нет необходимости стирать содержание старой функции `string_function()`, оно просто перезаписывается. Первый оператор вызывает функцию из математической библиотеки Lua, которая возвращает случайное число; параметры означают, что число должно быть целым от 65 до 90. Второй оператор

вызывает функцию из библиотеки Lua для работы со строками, которая преобразует число в символ; параметр представляет собой кодовую точку символа. К счастью, в кодировке ASCII символу „A“ соответствует значение 65, а „Z“ – 90, так что в результате всегда получим букву от A до Z.

For more about Lua math-library functions see Lua users «Math Library Tutorial». For more about Lua string-library functions see Lua users «String Library Tutorial» .

И снова функцию `string_function()` можно вызвать из main_function(), которую можно вызвать с помощью `main_function()`.

Теперь вывод на экране выглядит следующим образом:

```
tarantool> function string_function()
         >   local random_number
         >   local random_string
         >   random_number = math.random(65, 90)
         >   random_string = string.char(random_number)
         >   return random_string
         > end
---
...
tarantool> main_function()
---
- C
...
tarantool>
```

. . . На самом деле, вывод не всегда будет именно таким, поскольку функция `math.random()` вызывает случайные числа. Но для наглядности случайные значения в строке не важны.

### Изменение функции для возврата строки из десяти случайных букв

Сейчас стало понятно, как вызывать строки из одной случайной буквы, поэтому можно перейти к нашей цели – возврату строки из десяти букв с помощью конкатенации десяти строк из одной случайной буквы в цикле.

```
function string_function()
  local random_number
  local random_string
  random_string = ""
  for x = 1,10,1 do
    random_number = math.random(65, 90)
    random_string = random_string .. string.char(random_number)
  end
  return random_string
end
```

Слова «for x = 1,10,1» означают: “начать с x, равного 1, зацикливать до тех пор, пока x не будет равен 10, увеличивать x на 1 на каждом шаге цикла”. Символ «..» означает «конкатенацию», то есть добавление строки справа от знака «..» к строке слева от знака «..». Поскольку в начале определяется, что random_string (случайная_строка) представляет собой «» (пустую строку), в результате получим, что в random_string 10 случайных букв. И снова функцию `string_function()` можно вызвать из `main_function()`, которую можно вызвать с помощью `main_function()`.

Для получения подробной информации о циклах в языке Lua, см. Главу 4.3.4 «Числовой оператор for» в руководстве по языку Lua (chapter 4.3.4 «Numeric for»).

Теперь вывод на экране выглядит следующим образом:

```
tarantool> function string_function()
         >   local random_number
         >   local random_string
         >   random_string = ""
         >   for x = 1,10,1 do
         >     random_number = math.random(65, 90)
         >     random_string = random_string .. string.char(random_number)
         >   end
         >   return random_string
         > end
---
...
tarantool> main_function()
---
- 'ZUDJBHKEFM'
...
tarantool>
```

### Составление кортежа из числа и строки

Сейчас стало понятно, как создать строку из 10 случайных букв, поэтому можно создать кортеж, который будет содержать число и строку из 10 случайных букв, с помощью функции в Tarantool-библиотеке Lua-функций.

```
function main_function()
  local string_value, t
  string_value = string_function()
  t = box.tuple.new({1, string_value})
  return t
end
```

После этого, «t» будет представлять собой значение нового кортежа с двумя полями. Первое поле является числовым: «1». Второе поле представляет собой случайную строку. И снова функцию `string_function()` можно вызвать из `main_function()`, которую можно вызвать с помощью `main_function()`.

Для получения подробной информации о кортежах в Tarantool, см. раздел *Вложенный модуль box.tuple* руководства по Tarantool.

Теперь вывод на экране выглядит следующим образом:

```
tarantool> function main_function()
         > local string_value, t
         > string_value = string_function()
         > t = box.tuple.new({1, string_value})
         > return t
         > end
---
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> main_function()
---
- [1, 'PNPZPCOOKA']
...
tarantool>
```

### Изменение основной функции main_function для вставки кортежа в базу данных

Сейчас стало понятно, как создавать кортеж, который содержит число и строку из десяти случайных букв, поэтому осталось только поместить этот кортеж в спейс tester. Следует отметить, что tester – это первый спейс, определенный в песочнице, поэтому он представляет собой таблицу в базе данных.

```
function main_function()
  local string_value, t
  string_value = string_function()
  t = box.tuple.new({1,string_value})
  box.space.tester:replace(t)
end
```

Здесь новая строка – `box.space.tester:replace(t)`. Имя содержит слово „tester“, потому что вставка будет осуществляться в спейс tester. Второй параметр представляет собой значение в кортеже. Для абсолютной точности мы могли ввести команду `box.space.tester:insert(t)`, а не `box.space.tester:replace(t)`, но слово «replace» (заменить) означает “вставить, даже если уже существует кортеж, у которого значение первичного ключа совпадает”, и это облегчит повтор упражнения, даже если песочница не пуста. После того, как это будет выполнено, спейс tester будет содержать кортеж с двумя полями. Первое поле будет 1. Второе поле будет представлять собой строку из десяти случайных букв. И снова функцию `string_function()` можно вызвать из `main_function()`, которую можно вызвать с помощью `main_function()`. Но функция `main_function()` не может полностью отразить ситуацию, поскольку она не возвращает t, она только размещает t в базе данных. Чтобы убедиться, что произошла вставка, используем SELECT-запрос.

```
main_function()
        box.space.tester:select{1}
```

Для получения подробной информации о вызовах insert и replace в Tarantool, см. разделы *Вложенный модуль box.space*, *space_object:insert()* и *space_object:replace()* руководства по Tarantool.

Теперь вывод на экране выглядит следующим образом:

```
tarantool> function main_function()
         >   local string_value, t
         >   string_value = string_function()
         >   t = box.tuple.new({1,string_value})
         >   box.space.tester:replace(t)
         > end
---
...
tarantool> main_function()
---
...
tarantool> box.space.tester:select{1}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
- - [1, 'EUJYVEECIL']
...
tarantool>
```

### Изменение основной функции main_function для вставки миллиона кортежей в базу данных

Сейчас стало понятно, как вставить кортеж в базу данных, поэтому несложно догадаться, как можно увеличить масштаб: вместо того, чтобы вставлять значение 1 для первичного ключа, вставьте значение переменной от 1 до миллиона в цикле. Поскольку уже рассматривалось, как заводить цикл, это будет несложно. Мы лишь добавим небольшой штрих – функцию распределения во времени.

```
function main_function()
  local string_value, t
  for i = 1,1000000,1 do
    string_value = string_function()
    t = box.tuple.new({i,string_value})
    box.space.tester:replace(t)
  end
end
start_time = os.clock()
main_function()
end_time = os.clock()
'insert done in ' .. end_time - start_time .. ' seconds'
```

Стандартная Lua-функция os.clock() вернет время ЦП в секундах с момента начала программы. Таким образом, выводя start_time = number of seconds (время_начала = число секунд) прямо перед вставкой, а затем выводя end_time = number of seconds (время_окончания = число секунд) сразу после вставки, можно рассчитать (время_окончания - время_начала) = затраченное время в секундах. Отобразим это значение путем ввода в запрос без операторов, что приведет к тому, что Tarantool отправит значение на клиент, который выведет это значение. (Ответ Lua на C-функцию `printf()`, а именно `print()`, также сработает.)

Для получения подробной информации о функции `os.clock()` см. Главу 22.1 «Дата и время» в руководстве по языку Lua (chapter 22.1 «Date and Time»). Для получения подробной информации о функции print() см. Главу 5 «Функции» в руководстве по языку Lua (chapter 5 «Functions»).

И поскольку наступает кульминация – повторно введем окончательные варианты всех необходимых запросов: запрос, который создает `string_function()`, запрос, который создает `main_function()`, и запрос, который вызывает `main_function()`.

```
function string_function()
          local random_number
          local random_string
          random_string = ""
          for x = 1,10,1 do
            random_number = math.random(65, 90)
            random_string = random_string .. string.char(random_number)
          end
          return random_string
        end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
         function main_function()
           local string_value, t
           for i = 1,1000000,1 do
             string_value = string_function()
             t = box.tuple.new({i,string_value})
             box.space.tester:replace(t)
           end
         end
         start_time = os.clock()
         main_function()
         end_time = os.clock()
         'insert done in ' .. end_time - start_time .. ' seconds'
```

Теперь вывод на экране выглядит следующим образом:

```
tarantool> function string_function()
         >   local random_number
         >   local random_string
         >   random_string = ""
         >   for x = 1,10,1 do
         >     random_number = math.random(65, 90)
         >     random_string = random_string .. string.char(random_number)
         >   end
         >   return random_string
         > end
---
...
tarantool> function main_function()
         >   local string_value, t
         >   for i = 1,1000000,1 do
         >     string_value = string_function()
         >     t = box.tuple.new({i,string_value})
         >     box.space.tester:replace(t)
         >   end
         > end
---
...
tarantool> start_time = os.clock()
---
...
tarantool> main_function()
---
...
tarantool> end_time = os.clock()
---
...
tarantool> 'insert done in ' .. end_time - start_time .. ' seconds'
---
- insert done in 37.62 seconds
...
tarantool>
```

Итак, мы доказали, что возможности Lua-функций довольно многообразны (на самом деле, с помо-

щью хранимых процедур на языке Lua в Tarantool можно сделать больше, чем с помощью хранимых процедур в некоторых SQL СУБД), и несложно комбинировать функции Lua-библиотек и функции Tarantool-библиотек.

Также мы показали, что вставка миллиона кортежей заняла 37 секунд. Хостом выступил ноутбук с ОС Linux. А изменив значение *wal_mode* на „none“ перед запуском теста, можно уменьшить затраченное время до 4 секунд.

### Подсчет суммы по JSON-полям во всех кортежах

Задание по данному практикуму: “Предположим, что в каждом кортеже есть строка в формате JSON. В каждой строке есть числовое поле формата JSON. Для каждого кортежа необходимо найти значение числового поля и прибавить его к переменной „sum“ (сумма). В конце функция должна вернуть переменную „sum“.” Цель данного упражнения – получить опыт в прочтении и обработке кортежей одновременно.

```
1  json = require('json')
2  function sum_json_field(field_name)
3    local v, t, sum, field_value, is_valid_json, lua_table
4    sum = 0
5    for v, t in box.space.tester:pairs() do
6      is_valid_json, lua_table = pcall(json.decode, t[2])
7      if is_valid_json then
8        field_value = lua_table[field_name]
9        if type(field_value) == "number" then sum = sum + field_value end
10     end
11   end
12   return sum
13 end
```

**СТРОКА 3: ЗАЧЕМ НУЖЕН «LOCAL».** Эта строка объявляет все переменные, которые будут использоваться в функции. На самом деле, нет необходимости в начале объявлять все переменные, а в длинной функции лучше объявить переменные прямо перед их использованием. Фактически объявлять переменные вообще необязательно, но необъявленная переменная будет «глобальной». Это представляется нежелательным для всех переменных, объявленных в строке 1, поскольку все они используются только в рамках функции.

**СТРОКА 5: ЗАЧЕМ НУЖЕН «PAIRS()».** Наша задача – пройти по всем строкам, что можно сделать двумя способами: с помощью *box.space.space_object:pairs()* или с помощью `variable = select(...)` с указанием `for i, n, 1 do` *`некая-функция`*`(variable[i]) end`. Для данного примера мы предпочли использовать `pairs()`.

**СТРОКА 5: НАЧАЛО ОСНОВНОГО ЦИКЛА.** Всё внутри цикла «`for`» будет повторяться до тех пор, пока не кончатся индекс-ключи. На полученный кортеж можно сослаться с помощью переменной `t`.

**LINE 6: WHY «PCALL».** If we simply said `lua_table = json.decode(t[2]))`, then the function would abort with an error if it encountered something wrong with the JSON string - a missing colon, for example. By putting the function inside «`pcall`» (protected call), we're saying: we want to intercept that sort of error, so if there's a problem just set `is_valid_json = false` and we will know what to do about it later.

**СТРОКА 6: ЗНАЧЕНИЕ.** Функция *json.decode* означает декодирование JSON-строки, а параметр t[2] представляет собой ссылку на JSON-строку. Здесь есть заранее заданные значения, а мы предполагаем, что JSON-строка была вставлена во второе поле кортежа. Например, предположим, что кортеж выглядит следующим образом:

```
field[1]: 444
field[2]: '{"Hello": "world", "Quantity": 15}'
```

что означает, что первое поле кортежа, первичное поле, представляет собой число, а второе поле кортежа, JSON-строка, является строкой. Таким образом, значение оператора будет следующим: «декодировать `t[2]` (второе поле кортежа) как JSON-строку; если обнаружится ошибка, то указать `is_valid_json = false`; если ошибок нет, указать `is_valid_json = true` и `lua_table` = Lua-таблица, в которой находится декодированная строка».

**СТРОКА 8.** Наконец, мы готовы получить значение JSON-поля из Lua-таблицы, взятое из JSON-строки. Значение в field_name (имя_поля), которое является параметром всей функции, должно представлять собой JSON-поле. Например, в JSON-строке `'{"Hello": "world", "Quantity": 15}'` есть два JSON-поля: «Hello» и «Quantity». Если вся функция вызывается с помощью `sum_json_field("Quantity")`, тогда `field_value = lua_table[field_name]` (значение_поля = Lua_таблица[имя_поля]) по сути аналогично `field_value = lua_table["Quantity"]` или даже `field_value = lua_table.Quantity`. Итак, этими тремя способами можно ввести следующую команду: получить значение поля Quantity в Lua-таблице и поместить его в переменную `field_value`.

**СТРОКА 9: ЗАЧЕМ НУЖЕН «IF».** Предположим, что JSON-строка не содержит синтаксических ошибок, но JSON-поле не является числовым или вовсе отсутствует. В таком случае выполнение функции прервется при попытке прибавить значение к сумме. Если сначала проверить, `type(field_value) == "number"` (тип(значение_поля) == «число»), можно избежать прерывания функции. Если вы уверены, что база данных в идеальном состоянии, этот шаг можно пропустить.

И функция готова. Пора протестировать ее. Начинаем с пустой базы данных так же, как с песочницы в *упражнения в «Руководстве для начинающих»*,

```
-- если спейс tester остался от предыдущего задания, удалите его
box.space.tester:drop()
box.schema.space.create('tester')
box.space.tester:create_index('primary', {parts = {1, 'unsigned'}})
```

затем добавим несколько кортежей, где первое поле является числовым, а второе поле представляет собой строку.

```
box.space.tester:insert{444, '{"Item": "widget", "Quantity": 15}'}
box.space.tester:insert{445, '{"Item": "widget", "Quantity": 7}'}
box.space.tester:insert{446, '{"Item": "golf club", "Quantity": "sunshine"}'}
box.space.tester:insert{447, '{"Item": "waffle iron", "Quantit": 3}'}
```

Для целей практики здесь допущены ошибки. В «golf club» и «waffle iron» поля Quantity не являются числовыми, поэтому будут игнорироваться. Таким образом, итоговая сумма для полей Quantity в JSON-строках должна быть следующей: 15 + 7 = 22.

Вызовите функцию с помощью `sum_json_field("Quantity")`.

```
tarantool> sum_json_field("Quantity")
---
- 22
...
```

It works. We'll just leave, as exercises for future improvement, the possibility that the «hard coding» assumptions could be removed, that there might have to be an overflow check if some field values are huge, and that the function should contain a *yield* instruction if the count of tuples is huge.

### Индексированный поиск по шаблонам

Here is a generic function which takes a field identifier and a search pattern, and returns all tuples that match. * The field must be the first field of a TREE index. * The function will use Lua pattern matching, which allows «magic characters» in regular expressions. * The initial characters in the pattern, as far as the first magic character, will be used as an index search key. For each tuple that is found via the index, there will be a match of the whole pattern. * To be *cooperative*, the function should yield after every 10 tuples, unless there is a reason to delay yielding. With this function, we can take advantage of Tarantool's indexes for speed, and take advantage of Lua's pattern matching for flexibility. It does everything that an SQL *LIKE* search can do, and far more.

Прочитайте следующий Lua-код, чтобы понять, как он работает. Комментарии, которые начинаются с «СМ. ПРИМЕЧАНИЕ ... » ссылаются на подробные объяснения, приведенные ниже.

```
function indexed_pattern_search(space_name, field_no, pattern)
  -- СМ. ПРИМЕЧАНИЕ №1 "ПОИСК НУЖНОГО ИНДЕКСА"
  if (box.space[space_name] == nil) then
    print("Error: Failed to find the specified space")
    return nil
  end
  local index_no = -1
  for i=0,box.schema.INDEX_MAX,1 do
    if (box.space[space_name].index[i] == nil) then break end
    if (box.space[space_name].index[i].type == "TREE"
        and box.space[space_name].index[i].parts[1].fieldno == field_no
        and (box.space[space_name].index[i].parts[1].type == "scalar"
        or box.space[space_name].index[i].parts[1].type == "string")) then
      index_no = i
      break
    end
  end
  if (index_no == -1) then
    print("Error: Failed to find an appropriate index")
    return nil
  end
  -- СМ. ПРИМЕЧАНИЕ №2 "ПОЛУЧЕНИЕ КЛЮЧА ИНДЕКСНОГО ПОИСКА ИЗ ШАБЛОНА"
  local index_search_key = ""
  local index_search_key_length = 0
  local last_character = ""
  local c = ""
  local c2 = ""
  for i=1,string.len(pattern),1 do
    c = string.sub(pattern, i, i)
    if (last_character ~= "%") then
      if (c == '^' or c == "$" or c == "(" or c == ")" or c == "."
                   or c == "[" or c == "]" or c == "*" or c == "+"
                   or c == "-" or c == "?") then
        break
      end
      if (c == "%") then
        c2 = string.sub(pattern, i + 1, i + 1)
        if (string.match(c2, "%p") == nil) then break end
        index_search_key = index_search_key .. c2
      else
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        index_search_key = index_search_key .. c
      end
    end
    last_character = c
  end
  index_search_key_length = string.len(index_search_key)
  if (index_search_key_length < 3) then
    print("Error: index search key " .. index_search_key .. " is too short")
    return nil
  end
  -- СМ. ПРИМЕЧАНИЕ №3 "ВНЕШНИЙ ЦИКЛ: НАЧАЛО"
  local result_set = {}
  local number_of_tuples_in_result_set = 0
  local previous_tuple_field = ""
  while true do
    local number_of_tuples_since_last_yield = 0
    local is_time_for_a_yield = false
    -- СМ. ПРИМЕЧАНИЕ №4 "ВНУТРЕННИЙ ЦИКЛ: ИТЕРАТОР"
    for _,tuple in box.space[space_name].index[index_no]:
    pairs(index_search_key,{iterator = box.index.GE}) do
      -- СМ. ПРИМЕЧАНИЕ №5 "ВНУТРЕННИЙ ЦИКЛ: ПРЕРЫВАНИЕ, ЕСЛИ КЛЮЧ ИНДЕКСА СЛИШКОМ␣
↪БОЛЬШОЙ"
      if (string.sub(tuple[field_no], 1, index_search_key_length)
      > index_search_key) then
        break
      end
      -- СМ. ПРИМЕЧАНИЕ №6 "ВНУТРЕННИЙ ЦИКЛ: ПРЕРЫВАНИЕ ПОСЛЕ КАЖДЫХ ДЕСЯТИ КОРТЕЖЕЙ --␣
↪ВОЗМОЖНО"
      number_of_tuples_since_last_yield = number_of_tuples_since_last_yield + 1
      if (number_of_tuples_since_last_yield >= 10
          and tuple[field_no] ~= previous_tuple_field) then
        index_search_key = tuple[field_no]
        is_time_for_a_yield = true
        break
        end
      previous_tuple_field = tuple[field_no]
      -- СМ. ПРИМЕЧАНИЕ №7 "ВНУТРЕННИЙ ЦИКЛ: ДОБАВЛЕНИЕ В РЕЗУЛЬТАТ, ЕСЛИ ШАБЛОН СОВПАДЕТ
↪"
      if (string.match(tuple[field_no], pattern) ~= nil) then
        number_of_tuples_in_result_set = number_of_tuples_in_result_set + 1
        result_set[number_of_tuples_in_result_set] = tuple
      end
    end
    -- СМ. ПРИМЕЧАНИЕ №8 "ВНЕШНИЙ ЦИКЛ: ПРЕРЫВАНИЕ ИЛИ ПЕРЕДАЧА УПРАВЛЕНИЯ И ПРОДОЛЖЕНИЕ"
    if (is_time_for_a_yield ~= true) then
      break
    end
    require('fiber').yield()
  end
  return result_set
end
```

ПРИМЕЧАНИЕ №1 «ПОИСК НУЖНОГО ИНДЕКСА» Вызывающий клиент передал space_name

(имя_спейса – строка) и field_no (номер_поля – число). Требования следующие: (a) тип индекса должен быть «TREE», поскольку для других типов индекса (HASH, BITSET, RTREE) поиск с *итератором=GE* не вернет строки, упорядоченные по строковому значению; (b) field_no должен представлять собой первую часть индекса; (c) поле должно содержать строки, потому что для других типов данных (как «unsigned») шаблоны поиска не применяются; Если индекс не удовлетворяет этим требованиям, выдать сообщение об ошибке и вернуть нулевое значение nil.

ПРИМЕЧАНИЕ №2 «ПОЛУЧЕНИЕ КЛЮЧА ИНДЕКСНОГО ПОИСКА ИЗ ШАБЛОНА» Вызывающий клиент передал шаблон (строку). Ключом поиска по индексу являются символы в шаблоне до первого магического символа. Магические символы в Lua: % ^ $ ( ) . [ ] * + - ?. Например, если задан шаблон «ABC.E», точка будет магическим символом, и ключом поиска по индексу будет «ABC». Однако есть затруднение ... Если символ «%» будет идти следом за знаком препинания, этот знак препинания экранируется, поэтому следует убрать «%» из ключа поиска по индексу. Например, если задан шаблон «AB%$E», знак доллара экранируется, поэтому ключом поиска по индексу будет «AB$E». Наконец, есть проверка длины ключа поиска по индексу – не менее трех символов, причем это число выбрано произвольно, и даже ноль здесь подойдет, но по короткому ключу поиск займет длительное время.

NOTE #3 – «OUTER LOOP: INITIATE» The function's job is to return a result set, just as `box.space...select <box_space-select>` would. We will fill it within an outer loop that contains an inner loop. The outer loop's job is to execute the inner loop, and possibly *yield*, until the search ends. The inner loop's job is to find tuples via the index, and put them in the result set if they match the pattern.

ПРИМЕЧАНИЕ №4 «ВНУТРЕННИЙ ЦИКЛ: ИТЕРАТОР» Цикл for здесь использует pairs(), см. *обяснение, что такое итераторы*. Во внутреннем цикле будет локальная переменная под названием «tuple» (кортеж), которая содержит последний кортеж, обнаруженный в ходе поиска по индексу.

ПРИМЕЧАНИЕ №5 «ВНУТРЕННИЙ ЦИКЛ: ПРЕРЫВАНИЕ, ЕСЛИ КЛЮЧ ИНДЕКСА СЛИШКОМ БОЛЬШОЙ» Используется итератор GE (Greater or Equal - больше или равно), поэтому необходимо уточнить: если ключ поиска по индексу включает в себя N символов, то крайние N символов слева от найденного поля индекса не должны быть больше ключа поиска. Например, если ключом поиска является „ABC“, то „ABCDE“ потенциально подходит, а „ABD“ означает, что в дальнейшем совпадений не будет.

ПРИМЕЧАНИЕ №6 «ВНУТРЕННИЙ ЦИКЛ: ПРЕРЫВАНИЕ ПОСЛЕ КАЖДЫХ ДЕСЯТИ КОРТЕЖЕЙ – ВОЗМОЖНО» Эта часть кода предназначена для кооперативной многозадачности. Число 10 выбрано произвольно, и как правило, большее число также подойдет. Простое правило гласит: «после проверки 10 кортежей передать управление, а затем возобновить поиск (то есть снова выполнять внутренний цикл), начиная с последнего обнаруженного значения». Однако, если индекс не уникален, или в индексе более одного поля, можно получить дублирующиеся результаты, например, {«ABC»,1}, {«ABC», 2}, {«ABC», 3}» – и будет трудно решить, с какого кортежа «ABC» возобновлять поиск. Таким образом, если найденное поле индекса совпадает с предыдущим найденным полем индекса, цикл не прерывается.

ПРИМЕЧАНИЕ №7 «ВНУТРЕННИЙ ЦИКЛ: ДОБАВЛЕНИЕ В РЕЗУЛЬТАТ, ЕСЛИ ШАБЛОН СОВПАДЕТ» Сравнение найденного поля индекса с шаблоном. Например, предположим, что вызывающий клиент передает шаблон «ABC.E», и существует поле индекса, содержащее «ABCDE». В таком случае, начальный ключ поиска будет «ABC». Таким образом, кортеж, содержащий поле индекса с «ABCDE» будет обнаружен итератором, поскольку «ABCDE» > «ABC». В этом случае, string.match вернет значение, отличное от нулевого nil. В итоге, этот кортеж можно добавить в результирующий набор данных.

ПРИМЕЧАНИЕ №8 «ВНЕШНИЙ ЦИКЛ: ПРЕРЫВАНИЕ ИЛИ ПЕРЕДАЧА УПРАВЛЕНИЯ И ПРОДОЛЖЕНИЕ» Существуют три условия, которые вызовут прерывание из внутреннего цикла: (1) цикл for заканчивается закономерно, потому что отсутствуют ключи индекса, которые больше или равны ключу поиска по индексу, (2) ключ индекса слишком большой, как описано в ПРИМЕЧАНИИ №5, (3) пора передавать управление, как описано в ПРИМЕЧАНИИ №6. Если условие (1) или условие

(2) соблюдается, другие действия не требуются, и внешний цикл также заканчивается. Только в том случае, если справедливо условие (3), внешний цикл должен передать управление, а затем продолжить выполнение. Если он продолжит выполнение, то внутренний цикл – поиск с итератором – будет выполняться снова с новым значением для ключа поиска по индексу.

ПРИМЕР:

Запустите Tarantool, скопируйте и вставьте код для функции `indexed_pattern_search()` и попробуйте выполнить следующее:

```
box.space.t:drop()
box.schema.space.create('t')
box.space.t:create_index('primary',{})
box.space.t:create_index('secondary',{unique=false,parts={2,'string',3,'string'}})
box.space.t:insert{1,'A','a'}
box.space.t:insert{2,'AB',''}
box.space.t:insert{3,'ABC','a'}
box.space.t:insert{4,'ABCD',''}
box.space.t:insert{5,'ABCDE','a'}
box.space.t:insert{6,'ABCDE',''}
box.space.t:insert{7,'ABCDEF','a'}
box.space.t:insert{8,'ABCDF',''}
indexed_pattern_search("t", 2, "ABC.E.")
```

Получим следующий результат:

```
tarantool> indexed_pattern_search("t", 2, "ABC.E.")
---
- - [7, 'ABCDEF', 'a']
...
```

### 3.8.6 Практическое задание на C

#### Хранимые процедуры на языке C

Tarantool может вызывать код на языке C с помощью *модулей*, *ffi* или хранимых процедур на C. В данном практическом задании рассматривается только третий метод, хранимые процедуры на языке C. На самом деле, программы всегда представляют собой функции на языке C, но исторически сложилось так, что широко используется фраза «хранимая процедура».

Данное практическое задание могут выполнить те, у кого есть пакет программ для разработки Tarantool и компилятор языка программирования C. Оно состоит из пяти задач:

(1) *easy.c* – выводит «hello world»;

(2) *harder.c* – декодирует переданное значение параметра;

(3) *hardest.c* – использует API для языка C для вставки в базу данных;

(4) *read.c* – использует API для языка C для выборки из базы данных;

(5) *write.c* – использует API для языка C для замены в базе данных.

По окончании задания, вы увидите описанные здесь результаты и сможете самостоятельно написать хранимые процедуры.

### Preparation

Проверьте наличие следующих элементов на компьютере:

- Tarantool 2.1 or later
- Компилятор GCC, подойдет любая современная версия
- `module.h` и включенные в него файлы
- `msgpuck.h`
- `libmsgpuck.a` (только для некоторых последних версий msgpuck)

Файл `module.h` есть в системе, если Tarantool был установлен из исходных файлов. В противном случае, следует установить пакет Tarantool «developer». Например, на Ubuntu введите команду:

```
$ sudo apt-get install tarantool-dev
```

или на Fedora введите команду:

```
$ dnf -y install tarantool-devel
```

The `msgpuck.h` file will exist if Tarantool was installed from source. Otherwise the «msgpuck» package must be installed from https://github.com/tarantool/msgpuck.

Чтобы компилятор C увидел файлы `module.h` и `msgpuck.h`, путь к ним следует сохранить в переменной. Например, если адрес файла `module.h` – `/usr/local/include/tarantool/module.h`, а адрес файла `msgpuck.h` – `/usr/local/include/msgpuck/msgpuck.h`, введите команду:

```
$ export CPATH=/usr/local/include/tarantool:/usr/local/include/msgpuck
```

Статическая библиотека `libmsgpuck.a` нужна для версий msgpuck старше февраля 2017 года. Только в том случае, если встречаются проблемы соединения при использовании операторов GCC в примерах данного практического задания, в пути следует указывать `libmsgpuck.a` (`libmsgpuck.a` создан из исходных файлов загрузки msgpuck и Tarantool, поэтому его легко найти). Например, вместо «`gcc -shared -o harder.so -fPIC harder.c`» во втором примере ниже, необходимо ввести «`gcc -shared -o harder.so -fPIC harder.c libmsgpuck.a`».

Tarantool выполняет запросы в качестве *клиента*. Запустите Tarantool и введите эти запросы.

```
box.cfg{listen=3306}
box.schema.space.create('capi_test')
box.space.capi_test:create_index('primary')
net_box = require('net.box')
capi_connection = net_box:new(3306)
```

Проще говоря: создайте спейс под названием `capi_test`, и выполните соединение с одноименным `capi_connection`.

Не закрывайте клиент. Он понадобится для последующих запросов.

#### easy.c

Запустите еще один терминал. Измените директорию (cd), чтобы она совпадала с директорией, где запущен клиент.

Создайте файл. Назовите его `easy.c`. Запишите в него следующие шесть строк.

```
#include "module.h"
int easy(box_function_ctx_t *ctx, const char *args, const char *args_end)
{
  printf("hello world\n");
  return 0;
}
int easy2(box_function_ctx_t *ctx, const char *args, const char *args_end)
{
  printf("hello world -- easy2\n");
  return 0;
}
```

Скомпилируйте программу, что создаст файл библиотеки под названием `easy.so`:

```
$ gcc -shared -o easy.so -fPIC easy.c
```

Теперь вернитесь в клиент и выполните следующие запросы:

```
box.schema.func.create('easy', {language = 'C'})
box.schema.user.grant('guest', 'execute', 'function', 'easy')
capi_connection:call('easy')
```

Если эти запросы вам незнакомы, перечитайте описание *box.schema.func.create()*, *box.schema.user.grant()* и *conn:call()*.

Важна функция `capi_connection:call('easy')`.

Во-первых, она ищет функцию easy, что должно быть легко, потому что по умолчанию Tarantool ищет в текущей директории файл под названием `easy.so`.

Во-вторых, она вызывает функцию easy. Поскольку функция `easy()` в `easy.c` начинается с `printf("hello world\n")`, слова «hello world» появятся на экране.

В-третьих, она проверяет, что вызов прошел успешно. Поскольку функция `easy()` в `easy.c` оканчивается на `return 0`, сообщение об ошибке отсутствует, и запрос выполнен.

Результат должен выглядеть следующим образом:

```
tarantool> capi_connection:call('easy')
hello world
---
- []
...
```

Теперь вызовем другую функцию в easy.c – `easy2()`. Она практически совпадает с функцией `easy()`, но есть небольшое отличие: если имя файла не совпадет с именем функции, нужно будет указать *имя-файла*.*имя-функции*.

```
box.schema.func.create('easy.easy2', {language = 'C'})
box.schema.user.grant('guest', 'execute', 'function', 'easy.easy2')
capi_connection:call('easy.easy2')
```

. . . и на этот раз результатом будет: «hello world – easy2».

Вывод: вызвать C-функцию легко.

### harder.c

Вернитесь в терминал, где была создана программа `easy.c`.

Создайте файл. Назовите его `harder.c`. Запишите в него следующие 17 строк:

```
#include "module.h"
#include "msgpuck.h"
int harder(box_function_ctx_t *ctx, const char *args, const char *args_end)
{
  uint32_t arg_count = mp_decode_array(&args);
  printf("arg_count = %d\n", arg_count);
  uint32_t field_count = mp_decode_array(&args);
  printf("field_count = %d\n", field_count);
  uint32_t val;
  int i;
  for (i = 0; i < field_count; ++i)
  {
    val = mp_decode_uint(&args);
    printf("val=%d.\n", val);
  }
  return 0;
}
```

Скомпилируйте программу, что создаст файл библиотеки под названием `harder.so`:

```
$ gcc -shared -o harder.so -fPIC harder.c
```

Теперь вернитесь в клиент и выполните следующие запросы:

```
box.schema.func.create('harder', {language = 'C'})
box.schema.user.grant('guest', 'execute', 'function', 'harder')
passable_table = {}
table.insert(passable_table, 1)
table.insert(passable_table, 2)
table.insert(passable_table, 3)
capi_connection:call('harder', {passable_table})
```

На этот раз вызов передает Lua-таблицу (`passable_table`) в функцию `harder()`. Функция``harder()`` увидит это, как указано в параметре `char *args`.

At this point the `harder()` function will start using functions defined in msgpuck.h. The routines that begin with «mp» are msgpuck functions that handle data formatted according to the MsgPack specification. Passes and returns are always done with this format so one must become acquainted with msgpuck to become proficient with the C API.

Однако, пока достаточно понимать, что функция `mp_decode_array()` возвращает количество элементов в массиве, а функция `mp_decode_uint` возвращает целое число без знака из `args`. Есть также побочный эффект: по окончании декодирования `args` изменился и теперь указывает на следующий элемент.

Таким образом, первой будет отображена строка «arg_count = 1», поскольку был передан только один элемент: `passable_table`. Второй будет отображена строка «field_count = 3», потому что в таблице находятся три элемента. Следующие три строки будут «1», «2» и «3», потому что это значения элементов в таблице.

Теперь вывод на экране выглядит следующим образом:

```
tarantool> capi_connection:call('harder', passable_table)
arg_count = 1
field_count = 3
val=1.
val=2.
val=3.
---
- []
...
```

Вывод: на первый взгляд, декодирование значений параметров, переданных в C-функцию непросто, но существуют документированные процедуры для этих целей, и их не так много.

#### hardest.c

Вернитесь в терминал, где были созданы программы `easy.c` и `harder.c`.

Создайте файл. Назовите его `hardest.c. Запишите в него следующие 13 строк:

```
#include "module.h"
#include "msgpuck.h"
int hardest(box_function_ctx_t *ctx, const char *args, const char *args_end)
{
  uint32_t space_id = box_space_id_by_name("capi_test", strlen("capi_test"));
  char tuple[1024]; /* Must be big enough for mp_encode results */
  char *tuple_pointer = tuple;
  tuple_pointer = mp_encode_array(tuple_pointer, 2);
  tuple_pointer = mp_encode_uint(tuple_pointer, 10000);
  tuple_pointer = mp_encode_str(tuple_pointer, "String 2", 8);
  int n = box_insert(space_id, tuple, tuple_pointer, NULL);
  return n;
}
```

Скомпилируйте программу, что создаст файл библиотеки под названием `hardest.so`:

```
$ gcc -shared -o hardest.so -fPIC hardest.c
```

Теперь вернитесь в клиент и выполните следующие запросы:

```
box.schema.func.create('hardest', {language = "C"})
box.schema.user.grant('guest', 'execute', 'function', 'hardest')
box.schema.user.grant('guest', 'read,write', 'space', 'capi_test')
capi_connection:call('hardest')
```

На этот раз C-функция выполняет три действия:

(1) найдет числовой идентификатор спейса `capi_test` путем вызова `box_space_id_by_name()`;

(2) форматирует кортеж, используя другие функции `msgpuck.h`;

(3) вставит кортеж с помощью `box_insert()`.

**Предупреждение:** `char tuple[1024];` используется здесь просто в качестве быстрого способа ввода команды «выделить байтов с запасом». В серьезных программах разработчику следует обратить внимание на то, чтобы выделить достаточно места, которое будут использовать процедуры `mp_encode`.

Затем всё еще в клиенте выполните следующий запрос:

```
box.space.capi_test:select()
```

Результат должен выглядеть следующим образом:

```
tarantool> box.space.capi_test:select()
---
- - [10000, 'String 2']
...
```

Это доказывает, что функция `hardest()` была успешно выполнена, но откуда взялись *box_space_id_by_name()* и *box_insert()*? Ответ: *API для языка C*.

### read.c

Вернитесь в терминал, где были созданы программы `easy.c`, `harder.c` и `hardest.c`.

Создайте файл. Назовите его `read.c`. Запишите в него следующие 43 строки:

```
#include "module.h"
#include <msgpuck.h>
int read(box_function_ctx_t *ctx, const char *args, const char *args_end)
{
  char tuple_buf[1024];      /* where the raw MsgPack tuple will be stored */
  uint32_t space_id = box_space_id_by_name("capi_test", strlen("capi_test"));
  uint32_t index_id = 0;     /* The number of the space's first index */
  uint32_t key = 10000;      /* The key value that box_insert() used */
  mp_encode_array(tuple_buf, 0); /* clear */
  box_tuple_format_t *fmt = box_tuple_format_default();
  box_tuple_t *tuple = NULL;
  char key_buf[16];          /* Pass key_buf = encoded key = 1000 */
  char *key_end = key_buf;
  key_end = mp_encode_array(key_end, 1);
  key_end = mp_encode_uint(key_end, key);
  assert(key_end <= key_buf + sizeof(key_buf));
  /* Get the tuple. There's no box_select() but there's this. */
  int r = box_index_get(space_id, index_id, key_buf, key_end, &tuple);
  assert(r == 0);
  assert(tuple != NULL);
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  /* Get each field of the tuple + display what you get. */
  int field_no;             /* The first field number is 0. */
  for (field_no = 0; field_no < 2; ++field_no)
  {
    const char *field = box_tuple_field(tuple, field_no);
    assert(field != NULL);
    assert(mp_typeof(*field) == MP_STR || mp_typeof(*field) == MP_UINT);
    if (mp_typeof(*field) == MP_UINT)
    {
      uint32_t uint_value = mp_decode_uint(&field);
      printf("uint value=%u.\n", uint_value);
    }
    else /* if (mp_typeof(*field) == MP_STR) */
    {
      const char *str_value;
      uint32_t str_value_length;
      str_value = mp_decode_str(&field, &str_value_length);
      printf("string value=%.*s.\n", str_value_length, str_value);
    }
  }
  return 0;
}
```

Скомпилируйте программу, что создаст файл библиотеки под названием `read.so`:

```
$ gcc -shared -o read.so -fPIC read.c
```

Теперь вернитесь в клиент и выполните следующие запросы:

```
box.schema.func.create('read', {language = "C"})
box.schema.user.grant('guest', 'execute', 'function', 'read')
box.schema.user.grant('guest', 'read,write', 'space', 'capi_test')
capi_connection:call('read')
```

На этот раз C-функция выполняет четыре действия:

(1) снова найдет числовой идентификатор спейса `capi_test` путем вызова `box_space_id_by_name()`;

(2) форматирует ключ поиска = 10 000, используя другие функции `msgpuck.h`;

(3) получает кортеж с помощью `box_index_get()`;

(4) проходит по полям каждого кортежа с помощью `box_tuple_get()`. а затем декодирует каждое поле в зависимости от его типа. В данном случае, поскольку мы получаем кортеж, который сами вставили с помощью `hardest.c`, мы знаем заранее, что его тип будет MP_UINT или MP_STR. Однако, весьма часто здесь употребляется оператор выбора case с одной опцией для каждого возможного типа.

В результате вызова `capi_connection:call('read')` должны получить:

```
tarantool> capi_connection:call('read')
uint value=10000.
string value=String 2.
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- []
...
```

Это доказывает, что функция `read()` была успешно выполнена. И снова важные функции, которые начинаются с `box` – *box_index_get()* и *box_tuple_field()* – пришли из *API для языка C*.

### write.c

Вернитесь в терминал, где были созданы программы `easy.c`, `harder.c`, `hardest.c` и `read.c`.

Создайте файл. Назовите его `write.c`. Запишите в него следующие 24 строки:

```
#include "module.h"
#include <msgpuck.h>
int write(box_function_ctx_t *ctx, const char *args, const char *args_end)
{
  static const char *space = "capi_test";
  char tuple_buf[1024]; /* Должен быть достаточно большим, чтобы вместить результат mp_
↪encode */
  uint32_t space_id = box_space_id_by_name(space, strlen(space));
  if (space_id == BOX_ID_NIL) {
    return box_error_set(__FILE__, __LINE__, ER_PROC_C,
    "Can't find space %s", "capi_test");
  }
  char *tuple_end = tuple_buf;
  tuple_end = mp_encode_array(tuple_end, 2);
  tuple_end = mp_encode_uint(tuple_end, 1);
  tuple_end = mp_encode_uint(tuple_end, 22);
  box_txn_begin();
  if (box_replace(space_id, tuple_buf, tuple_end, NULL) != 0)
    return -1;
  box_txn_commit();
  fiber_sleep(0.001);
  struct tuple *tuple = box_tuple_new(box_tuple_format_default(),
                                      tuple_buf, tuple_end);
  return box_return_tuple(ctx, tuple);
}
```

Скомпилируйте программу, что создаст файл библиотеки под названием `write.so`:

```
$ gcc -shared -o write.so -fPIC write.c
```

Теперь вернитесь в клиент и выполните следующие запросы:

```
box.schema.func.create('write', {language = "C"})
box.schema.user.grant('guest', 'execute', 'function', 'write')
box.schema.user.grant('guest', 'read,write', 'space', 'capi_test')
capi_connection:call('write')
```

На этот раз C-функция выполняет шесть действий:

(1) снова найдет числовой идентификатор спейса `capi_test` путем вызова `box_space_id_by_name()`;

(2) создает новый кортеж;

(3) начинает транзакцию;

(4) заменяет кортеж в `box.space.capi_test`

(5) заканчивает транзакцию;

(6) последняя строка заменяет цикл `read.c` – вместо получения и вывода каждого поля, использует функцию `box_return_tuple(...)` для возврата всего кортежа вызывающему клиенту, чтобы вывести его на экран.

В результате вызова `capi_connection:call('write')` должны получить:

```
tarantool> capi_connection:call('write')
---
- [[1, 22]]
...
```

Это доказывает, что функция `write()` была успешно выполнена. И снова важные функции, которые начинаются с `box` – *box_txn_begin()*, *box_txn_commit()* и *box_return_tuple()* – пришли из *API для языка C*.

Вывод: длинное описание всего API для языка C необходимо в силу весомых причин. Все функции можно вызвать из C-функций, которые вызываются из Lua. Таким образом, хранимые процедуры на языке C получают полный доступ к базе данных.

#### Cleaning up

- Удалите все кортежи с функцией с помощью *box.schema.func.drop()*.
- Удалите спейс `capi_test` с помощью *box.schema.capi_test:drop()*.
- Удалите файлы с разрешением `.c` и `.so`, созданные для данного практического задания.

#### An example in the test suite

Скачайте исходный код Tarantool. Откройте поддиректорию `test/box`. Проверьте наличие файла под названием `tuple_bench.test.lua` и еще одного файла под названием `tuple_bench.c`. Изучите Lua-файл на предмет вызова функции в C-файле с использованием методов, описанных в данном практическом задании.

Вывод: некоторые тесты из стандартного набора используют хранимые процедуры на языке C, а они должны работать, поскольку мы не можем выпустить Tarantool, если он не прошел тестирование.

## 3.9 SQL guides

This section contains hands-on SQL guides. You might also want to read the in-depth *SQL reference*.

### 3.9.1 Руководство для начинающих по SQL

В этом руководстве описано, как пользователю начать работу с SQL в Tarantool, а также объясняются все необходимые понятия.

Ниже рассказывается о базах данных в целом и взаимосвязи между решениями NoSQL и SQL в Tarantool. Большинство тем, изложенных в этом руководстве, будут знакомы тем, кто уже использовал реляционные базы данных.

#### Sample table

В футбольном тренировочном лагере тренер традиционно начинает с того, что показывает игрокам футбольный мяч и говорит: «Это — футбольный мяч». В случае SQL подобный базовый объект — это таблица:

```
TABLE
          [1]              [2]              [3]
       +----------------+----------------+----------------+
 Стрк1 |  Стрк1, Стлб1   |  Стрк1, Стлб2  |  Стрк1, Стлб3  |
       +----------------+----------------+----------------+
 Стрк2 |  Стрк2, Стлб1   | Стрк2, Стлб2   |  Стрк2, Стлб3  |
       +----------------+----------------+----------------+
 Стрк3 |  Стрк3, Стлб1   |  Стрк3, Стлб2  |  Стрк3, Стлб3  |
       +----------------+----------------+----------------+
```

But the labels are misleading – one usually doesn't identify rows and columns by their ordinal positions, one prefers to pick out specific items by their contents. In that spirit, this is a table:

```
MODULES

+-----------------+------+---------------------+
| NAME            | SIZE | PURPOSE             |
+-----------------+------+---------------------+
| box             | 1432 | Управление БД       |
| clock           |  188 | Секунды             |
| crypto          |    4 | Криптография        |
+-----------------+------+---------------------+
```

So one does not use longitude/latitude navigation by talking about «Row#2 Column #2», one uses the contents of the Name column and the name of the Size column by talking about «the size, where the name is „clock“». To be more exact, this is what one says:

```
SELECT size FROM modules WHERE name = 'clock';
```

If you're familiar with Tarantool's architecture – and ideally you read about that before coming to this chapter – then you know that there is a NoSQL way to get the same thing:

```
box.space.MODULES:select()[2][2]
```

Well, you can do that. One of the advantages of Tarantool is that if you can get data via an SQL statement, then you can get the same data via a NoSQL request. But the reverse is not true, because not all NoSQL tuple sets are definable as SQL tables. These restrictions apply for SQL that do not apply for NoSQL: 1. Every column must have a name. 2. Every column should have a scalar type (Tarantool is relaxed about which particular scalar type you can have, but there is no way to index and search arrays, tables within tables, or what MessagePack calls «maps».)

Tarantool/NoSQL's «format» clause causes the same restrictions.

So an SQL «table» is a NoSQL «tuple set with format restrictions», an SQL «row» is a NoSQL «tuple», an SQL «column» is a NoSQL «list of fields within a tuple set».

### Creating a table

This is how to create the modules table:

```
CREATE TABLE modules (name STRING, size INTEGER, purpose STRING, PRIMARY KEY (name));
```

The words that are IN CAPITAL LETTERS are «keywords» (although it is only a convention in this manual that keywords are in capital letters, in practice many programmers prefer to avoid shouting). A keyword has meaning for the SQL parser so many keywords are reserved, they cannot be used as names unless they are enclosed inside quotation marks.

The word «modules» is a «table name», and the words «name» and «size» and «purpose» are «column names». All tables and all columns must have names.

The words «STRING» and «INTEGER» are «data types». STRING means «the contents should be characters, the length is indefinite, the equivalent NoSQL type is „string““». INTEGER means «the contents should be numbers without decimal points, the equivalent NoSQL type is „integer“». Tarantool supports other data types but this section's example table has data types from the two main groups, namely, data types for numbers and data types for strings.

The final clause, PRIMARY KEY (name), means that the name column is the main column used to identify the row.

### Nulls

Frequently it is necessary, at least temporarily, that a column value should be NULL. Typical situations are: the value is unknown, or the value is not applicable. For example, you might make a module as a placeholder but you don't want to say its size or purpose. If such things are possible, the column is «nullable». The example table's name column cannot contain nulls, and it could be defined explicitly as «name STRING NOT NULL», but in this case that's unnecessary – a column defined as PRIMARY KEY is automatically NOT NULL.

Is a NULL in SQL the same thing as a nil in Lua? No, but it is close enough that there will be confusion. When nil means «unknown» or «inapplicable», yes. But when nil means «nonexistent» or «type is nil», no. NULL is a value, it has a data type because it is inside a column which is defined with that data type.

### Creating an index

This is how to create indexes for the modules table:

```
CREATE INDEX size ON modules (size);
CREATE UNIQUE INDEX purpose ON modules (purpose);
```

There is no need to create an index on the name column, because Tarantool creates an index automatically when it sees a PRIMARY KEY clause in the CREATE TABLE statement. In fact there is no need to create indexes on the size or purpose columns either – if indexes don't exist, then it is still possible to use the columns for searches. Typically people create non-primary indexes, also called secondary indexes, when it becomes clear that the table will grow large and searches will be frequent, because searching with an index is generally much faster than searching without an index.

Another use for indexes is to enforce uniqueness. When an index is created with CREATE UNIQUE INDEX for the purpose column, it is not possible to have duplicate values in that column.

### Data change

Putting data into a table is called «inserting». Changing data is called «updating». Removing data is called «deleting». Together, the three SQL statements INSERT plus UPDATE plus DELETE are the three main «data-change» statements.

This is how to insert, update, and delete a row in the modules table:

```
INSERT INTO modules VALUES ('json', 14, 'format functions for JSON');
UPDATE modules SET size = 15 WHERE name = 'json';
DELETE FROM modules WHERE name = 'json';
```

The corresponding non-SQL Tarantool requests would be:

```
box.space.MODULES:insert{'json', 14, 'format functions for JSON'}
box.space.MODULES:update('json', {{'=', 2, 15}})
box.space.MODULES:delete{'json'}
```

This is how one would populate the table with the values that was shown earlier:

```
INSERT INTO modules VALUES ('box', 1432, 'Database Management');
INSERT INTO modules VALUES ('clock', 188, 'Seconds');
INSERT INTO modules VALUES ('crypto', 4, 'Cryptography');
```

### Constraints

Some data-change statements are illegal due to something in the table's definition. This is called «constraining what can be done». Some types of constraints have already been shown . . .

NOT NULL – if a column is defined with a NOT NULL clause, it is illegal to put NULL into it. A primary-key column is automatically NOT NULL.

UNIQUE – if a column has a UNIQUE index, it is illegal to put a duplicate into it. A primary-key column automatically has a UNIQUE index.

Допустимые значения — если столбец по определению содержит тип данных INTEGER, в него запрещено помещать нечисловые значения. В целом, недопустимо любое значение, которое не соответствует типу данных, указанному в определении. Некоторые системы управления базами данных относительно гибки и скорее примут некорректные значения, чем отклонят их. Tarantool в этом плане более строгий.

Now, here are other types of constraints . . .

CHECK – a table description can have a clause «CHECK (conditional expression)». For example, if the CREATE TABLE modules statement looked like this:

```
CREATE TABLE modules (name STRING,
                      size INTEGER,
                      purpose STRING,
                      PRIMARY KEY (name),
                      CHECK (size > 0));
```

then this INSERT statement would be illegal: `INSERT INTO modules VALUES ('box', 0, 'The Database Kernel');` because there is a CHECK constraint saying that the second column, the size column, cannot

contain a value which is less than or equal to zero. Try this instead: `INSERT INTO modules VALUES ('box', 1, 'The Database Kernel');`

FOREIGN KEY – a table description can have a clause «FOREIGN KEY (column-list) REFERENCES table (column-list)». For example, if there is a new table «submodules» which in a way depends on the modules table, it can be defined like this:

```
CREATE TABLE submodules (name STRING,
                         module_name STRING,
                         size INTEGER,
                         purpose STRING,
                         PRIMARY KEY (name),
                         FOREIGN KEY (module_name) REFERENCES
                         modules (name));
```

Now try to insert a new row into this submodules table:

```
INSERT INTO submodules VALUES
  ('space', 'Box', 10000, 'insert etc.');
```

The insert will fail because the second column (module_name) refers to the name column in the modules table, and the name column in the modules table does not contain „Box“. However, it does contain „box“. By default searches in Tarantool's SQL use a binary collation. This will work:

```
INSERT INTO submodules
  VALUES ('space', 'box', 10000, 'insert etc.');
```

Теперь попробуем удалить соответствующую строку из таблицы modules:

```
DELETE FROM modules WHERE name = 'box';
```

The delete will fail because the second column (module_name) in the submodules table refers to the name column in the modules table, and the name column in the modules table would not contain „box“ if the delete succeeded. So the FOREIGN KEY constraint affects both the table which contains the FOREIGN KEY clause and the table that the FOREIGN KEY clause refers to.

The constraints in a table's definition – NOT NULL, UNIQUE, data domain, CHECK, and FOREIGN KEY – are guarantors of the database's integrity. It is important that they are fixed and well-defined parts of the definition, and hard to bypass with SQL. This is often seen as a difference between SQL and NoSQL – SQL emphasizes law and order, NoSQL emphasizes freedom and making your own rules.

### Table relationships

Think about the two tables that have been discussed so far:

```
CREATE TABLE modules (name STRING,
                      size INTEGER,
                       purpose STRING,
                       PRIMARY KEY (name),
                       CHECK (size > 0));

CREATE TABLE submodules (name STRING,
                         module_name STRING,
                         size INTEGER,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                        purpose STRING,
                        PRIMARY KEY (name),
                        FOREIGN KEY (module_name) REFERENCES
                        modules (name));
```

Because of the FOREIGN KEYS clause in the submodules table, there is clearly a many-to-one relationship: submodules –>> modules that is, every submodules row must refer to one (and only one) modules row, while every modules row can be referred to in zero or more submodules rows.

Table relationships are important, but beware: do not trust anyone who tells you that databases made with SQL are relational «because there are relationships between tables». That is wrong, as will be clear in the discussion about what makes a database relational, later.

### Selecting with WHERE

---

**Важно:** By default, Tarantool prohibits `SELECT` queries that scan table rows instead of using indexes to avoid unwanted heavy load. For the purposes of this tutorial, allow SQL scan queries in Tarantool by running the command:

```
SET SESSION "sql_seq_scan" = true;
```

Alternatively, you can allow a specific query to perform a table scan by adding the `SEQSCAN` keyword before the table name. Learn more about using `SEQSCAN` in SQL scan queries in the *SQL FROM clause description*.

---

We gave a simple example of a SELECT statement earlier:

```
SELECT size FROM modules WHERE name = 'clock';
```

The clause «WHERE name = „clock“» is legal in other statements – it is in examples with UPDATE and DELETE – but here the only examples will be with SELECT.

The first variation is that the WHERE clause does not have to be specified at all, it is optional. So this statement would return all rows:

```
SELECT size FROM modules;
```

The second variation is that the comparison operator does not have to be „=“, it can be anything that makes sense: „>“ or „>=“ or „<“ or „<=“, or „LIKE“ which is an operator that works with strings that may contain wildcard characters „_“ meaning „match any one character“ or „%“ meaning „match any zero or one or many characters“. These are legal statements which return all rows:

```
SELECT size FROM modules WHERE name >= '';
SELECT size FROM modules WHERE name LIKE '%';
```

The third variation is that IS [NOT] NULL is a special condition. Remembering that the NULL value can mean «it is unknown what the value should be», and supposing that in some row the size is NULL, then the condition «size > 10» is not certainly true and it is not certainly false, so it is evaluated as «unknown». Ordinarily the application of a WHERE clause filters out both false and unknown results. So when searching for NULL, say IS NULL; when searching anything that is not NULL, say IS NOT NULL. This statement will return all rows because (due to the definition) there are no NULLs in the name column:

```
SELECT size FROM modules WHERE name IS NOT NULL;
```

The fourth variation is that conditions can be combined with AND / OR, and negated with NOT.

So this statement would return all rows (the first condition is false but the second condition is true, and OR means «return true if either condition is true»):

```
SELECT size
FROM modules
WHERE name = 'wombat' OR size IS NOT NULL;
```

### Selecting with a select list

Yet again, here is a simple example of a SELECT statement:

```
SELECT size FROM modules WHERE name = 'clock';
```

The words between SELECT and FROM are the select list. In this case, the select list is just one word: size. Formally it means that the desire is to return the size values, and technically the name for picking a particular column is called «projection».

The first variation is that one can specify any column in any order:

```
SELECT name, purpose, size FROM modules;
```

The second variation is that one can specify an expression, it does not have to be a column name, it does not even have to include a column name. The common expression operators for numbers are the arithmetic operators + - / *; the common expression operator for strings is the concatenation operator ||. For example this statement will return 8, „XY“:

```
SELECT size * 2, 'X' || 'Y' FROM modules WHERE size = 4;
```

The third variation is that one can add a clause [AS name] after every expression, so that in the return the column titles will make sense. This is especially important when a title might otherwise be ambiguous or meaningless. For example this statement will return 8, „XY“ as before

```
SELECT size * 2 AS double_size, 'X' || 'Y' AS concatenated_literals  FROM modules
  WHERE size = 4;
```

but displayed as a table the result will look like

```
+----------------+------------------------+
| DOUBLE_SIZE    | CONCATENATED_LITERALS  |
+----------------+------------------------+
|              8 | XY                     |
+----------------+------------------------+
```

### Selecting with a select list with asterisk

Instead of listing columns in a select list, one can just say '*'. For example

```
SELECT * FROM modules;
```

This is the same thing as

```
SELECT name, size, purpose FROM modules;
```

Selecting with "*" saves time for the writer, but it is unclear to a reader who has not memorized what the column names are. Also it is unstable, because there is a way to change a table's definition (the ALTER statement, which is an advanced topic). Nevertheless, although it might be bad to use it for production, it is handy to use it for introduction, so "*" will appear in some following examples.

### Select with subqueries

Remember that there is a modules table and there is a submodules table. Suppose that there is a desire to list the submodules that refer to modules for which the purpose is X. That is, this involves a search of one table using a value in another table. This can be done by enclosing «(SELECT ...)» within the WHERE clause. For example:

```
SELECT name FROM submodules
WHERE module_name =
    (SELECT name FROM modules WHERE purpose LIKE '%Database%');
```

Subqueries are also useful in the select list, when one wishes to combine information from more than one table. For example this statement will display submodules rows but will include values that come from the modules table:

```
SELECT name AS submodules_name,
    (SELECT purpose FROM modules
     WHERE modules.name = submodules.module_name)
     AS modules_purpose,
    purpose AS submodules_purpose
FROM submodules;
```

Whoa. What are «modules.name» and «submodules.name»? Whenever you see «x . y» you are looking at a «qualified column name», and the first part is a table identifier, the second part is a column identifier. It is always legal to use qualified column names, but until now it has not been necessary. Now it is necessary, or at least it is a good idea, because both tables have a column named «name».

Результат будет выглядеть следующим образом:

```
+------------------+------------------------+--------------------+
| SUBMODULES_NAME  | MODULES_PURPOSE        | SUBMODULES_PURPOSE |
+------------------+------------------------+--------------------+
| space            | Управление БД          | Вставка ...        |
+------------------+------------------------+--------------------+
```

Perhaps you have read somewhere that SQL stands for «Structured Query Language». That is not true any more. But it is true that the query syntax allows for a structural component, namely the subquery, and that was the original idea. However, there is a different way to combine tables – with joins instead of subqueries.

### Select with Cartesian join

Until now only «FROM modules» or «FROM submodules» was used in SELECT statements. What if there was more than one table in the FROM clause? For example

```
SELECT * FROM modules, submodules;
```

or

```
SELECT * FROM modules JOIN submodules;
```

That is legal. Usually it is not what you want, but it is a learning aid. The result will be:

```
{ столбцы таблицы modules }         { столбцы таблицы submodules }
+--------+------+---------------------+-------+-------------+-------+-------------+
| NAME   | SIZE | PURPOSE             | NAME  | MODULE_NAME | SIZE  | PURPOSE     |
+--------+------+---------------------+-------+-------------+-------+-------------+
| box    | 1432 | Управление БД       | space | box         | 10000 | Вставка ... |
| clock  |  188 | Секунды             | space | box         | 10000 | Вставка ... |
| crypto |    4 | Криптография        | space | box         | 10000 | Вставка ... |
+--------+------+---------------------+-------+-------------+-------+-------------+
```

It is not an error. The meaning of this type of join is «combine every row in table-1 with every row in table-2». It did not specify what the relationship should be, so the result has everything, even when the submodule has nothing to do with the module.

It is handy to look at the above result, called a «Cartesian join» result, to see what would really be desirable. Probably for this case the row that actually makes sense is the one where the modules.name = submodules.module_name, and it's better to make that clear in both the select list and the WHERE clause, thus:

```
SELECT modules.name AS modules_name,
       modules.size AS modules_size,
       modules.purpose AS modules_purpose,
       submodules.name,
       module_name,
       submodules.size,
       submodules.purpose
FROM modules, submodules
WHERE modules.name = submodules.module_name;
```

Мы получим следующий результат:

```
+----------+-----------+------------+--------+---------+-------+-------------+
| MODULES_ |  MODULES_ | MODULES_   | NAME   | MODULE_ | SIZE  | PURPOSE     |
| NAME     |  SIZE     | PURPOSE    |        | NAME    |       |             |
+----------+-----------+--------- --+--------+---------+-------+-------------+
| box      |      1432 | Управление | space  | box     | 10000 | Вставка ... |
|          |           | БД         |        |         |       |             |
+----------+-----------+------------+--------+---------+-------+-------------+
```

In other words, you can specify a Cartesian join in the FROM clause, then you can filter out the irrelevant rows in the WHERE clause, and then you can rename columns in the select list. This is fine, and every SQL DBMS supports this. But it is worrisome that the number of rows in a Cartesian join is always (number of rows in first table multiplied by number of rows in second table), which means that conceptually you are often filtering in a large set of rows.

It is good to start by looking at Cartesian joins because they show the concept. Many people, though, prefer to use different syntaxes for joins because they look better or clearer. So now those alternatives will be shown.

### Select with join with ON clause

The ON clause would have the same comparisons as the WHERE clause that was illustrated for the previous section, but the use of different syntax would be making it clear «this is for the sake of the join». Readers can see at a glance that it is, in concept at least, an initial step before the result rows are filtered. For example this

```
SELECT * FROM modules JOIN submodules
  ON (modules.name = submodules.module_name);
```

is the same as

```
SELECT * FROM modules, submodules
  WHERE modules.name = submodules.module_name;
```

### Select with join with USING clause

The USING clause would take advantage of names that are held in common between the two tables, with the assumption that the intent is to match those columns with „=“ comparisons. For example,

```
SELECT * FROM modules JOIN submodules USING (name);
```

has the same effect as

```
SELECT * FROM modules JOIN submodules WHERE modules.name = submodules.name;
```

If the table had been created with a plan in advance to use USING clauses, that would save time. But that did not happen. So, although the above example «works», the results will not be sensible.

### Select with natural join

A natural join would take advantage of names that are held in common between the two tables, and would do the filtering automatically based on that knowledge, and throw away duplicate columns.

If the table had been created with a plan in advance to use natural joins, that would be very handy. But that did not happen. So, although the following example «works», the results won't be sensible.

```
SELECT * FROM modules NATURAL JOIN submodules;
```

Result: nothing, because modules.name does not match submodules.name, and so on And even if there had been a result, it would only have included four columns: name, module_name, size, purpose.

### Select with left join

Now what if there is a desire to join modules to submodules, but it's necessary to be sure that all the modules are found? In other words, suppose the requirement is to get modules even if the condition submodules.module_name = modules.name is not true, because the module has no submodules.

When that is the requirement, the type of join is an «outer join» (as opposed to the type that has been used so far which is an «inner join»). Specifically the format will be LEFT [OUTER] JOIN because the main table, modules, is on the left. For example:

```
SELECT *
FROM modules LEFT JOIN submodules
ON modules.name = submodules.module_name;
```

which returns:

```
{ столбцы таблицы modules    }         { столбцы таблицы submodules    }
+--------+------+---------------------+-------+-------------+-------+-------------+
| NAME   | SIZE | PURPOSE             | NAME  | MODULE_NAME | SIZE  | PURPOSE     |
+--------+------+---------------------+-------+-------------+-------+-------------+
| box    | 1432 | Управление БД       | space | box         | 10000 | Вставка ... |
| clock  |  188 | Секунды             | NULL  | NULL        | NULL  | NULL        |
| crypto |    4 | Криптография        | NULL  | NULL        | NULL  | NULL        |
+--------+------+---------------------+-------+-------------+-------+-------------+
```

Thus, for the submodules of the clock module and the submodules of the crypto module – which do not exist – there are NULLs in every column.

### Select with functions

A function can take any expression, including an expression that contains another function, and return a scalar value. There are many such functions. Here will be a description of only one, SUBSTR, which returns a substring of a string.

Format: `SUBSTR(`*`input-string`*`, `*`start-with`*` [, `*`length`*`])`

Description: SUBSTR takes input-string, eliminates any characters before start-with, eliminates any characters after (start-with plus length), and returns the result.

Example: `SUBSTR('abcdef', 2, 3)` returns „bcd“.

Select with aggregation, GROUP BY, and HAVING

Remember that the modules table looks like this:

```
MODULES

+-----------------+------+---------------------+
| NAME            | SIZE | PURPOSE             |
+-----------------+------+---------------------+
| box             | 1432 | Управление БД       |
| clock           |  188 | Секунды             |
| crypto          |    4 | Криптография        |
+-----------------+------+---------------------+
```

Suppose that there is no need to know all the individual size values, all that is important is their aggregation, that is, take the attributes of the collection. SQL allows aggregation functions including: AVG (average), SUM, MIN (minimum), MAX (maximum), and COUNT. For example

```
SELECT AVG(size), SUM(size), MIN(size), MAX(size), COUNT(size) FROM modules;
```

Результат будет выглядеть следующим образом:

```
+-----------+-----------+-----------+-----------+-----------+
| СТОЛБЕЦ1  | СТОЛБЕЦ2  | СТОЛБЕЦ3  | СТОЛБЕЦ4  | СТОЛБЕЦ5  |
+-----------+-----------+-----------+-----------+-----------|
|       541 |      1624 |         4 |      1432 |         3 |
+-----------+-----------+-----------+-----------+-----------+
```

Suppose that the requirement is aggregations, but aggregations of rows that have some common characteristic. Supposing further, the rows should be divided into two groups, the ones whose names begin with „b“ and the ones whose names begin with „c“. This can be done by adding a clause [GROUP BY expression]. For example,

```
SELECT SUBSTR(name, 1, 1), AVG(size), SUM(size), MIN(size), MAX(size), COUNT(size)
FROM modules
GROUP BY SUBSTR(name, 1, 1);
```

Результат будет выглядеть следующим образом:

```
+------------+--------------+-----------+-----------+-----------+-------------+
| СТОЛБЕЦ1   | СТОЛБЕЦ2     | СТОЛБЕЦ3  | СТОЛБЕЦ4  | СТОЛБЕЦ5  | СТОЛБЕЦ6    |
+------------+--------------+-----------+-----------+-----------+-------------+
| b          |         1432 |      1432 |      1432 |      1432 |           1 |
| c          |           96 |       192 |         4 |       188 |           2 |
+------------+--------------+-----------+-----------+-----------+-------------+
```

### Select with common table expression

It is possible to define a temporary (viewed) table within a statement, usually within a SELECT statement, using a WITH clause. For example:

```
WITH tmp_table AS (SELECT x1 FROM t1) SELECT * FROM tmp_table;
```

### Select with order, limit, and offset clauses

So far, tor every search in the modules table, the rows have come out in alphabetical order by name: „box“, then „clock“, then „crypto“. However, to really be sure about the order, or to ask for a different order, it is necessary to be explicit and add a clause: `ORDER BY column-name [ASC|DESC]`. (ASC stands for ASCending, DESC stands for DESCending.) For example:

```
SELECT * FROM modules ORDER BY name DESC;
```

The result will be the usual rows, in descending alphabetical order: „crypto“ then „clock“ then „box“.

After the ORDER BY clause there can be a clause LIMIT n, where n is the maximum number of rows to retrieve. For example:

```
SELECT * FROM modules ORDER BY name DESC LIMIT 2;
```

The result will be the first two rows, „crypto“ and „clock“.

After the ORDER BY clause and the LIMIT clause there can be a clause OFFSET n, where n is the row to start with. The first offset is 0. For example:

```
SELECT * FROM modules ORDER BY name DESC LIMIT 2 OFFSET 2;
```

The result will be the third row, „box“.

### Views

A view is a canned SELECT. If you have a complex SELECT that you want to run frequently, create a view and then do a simple SELECT on the view. For example:

```
CREATE VIEW v AS SELECT size, (size *5) AS size_times_5
FROM modules
GROUP BY size, name
ORDER BY size_times_5;
SELECT * FROM v;
```

### Transactions

Tarantool has a «Write Ahead Log» (WAL). Effects of data-change statements are logged before they are permanently stored on disk. This is a reason that, although entire databases can be stored in temporary memory, they are not vulnerable in case of power failure.

Tarantool supports commits and rollbacks. In effect, asking for a commit means asking for all the recent data-change statements, since a transaction began, to become permanent. In effect, asking for a rollback means asking for all the recent data-change statements, since a transaction began, to be cancelled.

For example, consider these statements:

```
CREATE TABLE things (remark STRING, PRIMARY KEY (remark));
START TRANSACTION;
INSERT INTO things VALUES ('A');
COMMIT;
START TRANSACTION;
INSERT INTO things VALUES ('B');
ROLLBACK;
SELECT * FROM things;
```

The result will be: one row, containing „A“. The ROLLBACK cancelled the second INSERT statement, but did not cancel the first one, because it had already been committed.

Ordinarily every statement is automatically committed.

After START TRANSACTION, statements are not automatically committed – Tarantool considers that a transaction is now «active», until the transaction ends with a COMMIT statement or a ROLLBACK statement. While a transaction is active, all statements are legal except another START TRANSACTION.

### Implementing Tarantool's SQL On Top of NoSQL

Tarantool's SQL data is the same as Tarantool's NoSQL data. When you create a table or an index with SQL, you are creating a space or an index in NoSQL. For example:

```
CREATE TABLE things (remark STRING, PRIMARY KEY (remark));
INSERT INTO things VALUES ('X');
```

is somewhat similar to

```
box.schema.space.create('THINGS',
{
    format = {
               [1] = {["name"] = "REMARK", ["type"] = "string"}
               }
})
box.space.THINGS:create_index('pk_unnamed_THINGS_1',{unique=true,parts={1,'string'}})
box.space.THINGS:insert{'X'}
```

Therefore you can take advantage of Tarantool's NoSQL features even though your primary language is SQL. Here are some possibilities.

(1) NoSQL applications written in one of the connector languages may be slightly faster than SQL applications because SQL statements may require more parsing and may be translated to NoSQL requests.

(2) You can write stored procedures in Lua, combining Lua loop-control and Lua library-access statements with SQL statements. These routines are executed on the server, which is the principal advantage of pure-SQL stored procedures.

(3) There are some options that are implemented in NoSQL that are not (yet) implemented in SQL. For example you can use NoSQL to change an index option, and to deny access to users named „guest".

(4) System spaces such as _space and _index can be accessed with SQL SELECT statements. This is not quite the same as an information_schema, but it does mean that you can use SQL to access the database's metadata catalog.

Fields in NoSQL spaces can be accessed with SQL if and only if they are scalar and are defined in format clauses. Indexes of NoSQL spaces will be used with SQL if and only if they are TREE indexes.

### Relational databases

Edgar F. Codd, the person most responsible for researching and explaining relational database concepts, listed the main criteria as (Codd's 12 rules).

Although Tarantool is not advertised as «relational», Tarantool comes with a claim that it complies with these rules, with the following caveats and exceptions ...

The rules state that all data must be viewable as relations. A Tarantool SQL table is a relation. However, it is possible to have duplicate values in SQL tables and it is possible to have an implicit ordering. Those characteristics are not allowed for true relations.

The rules state that there must be a dynamic online catalog. Tarantool has one but some metadata is missing from it.

The rules state that the data language must support authorization. Tarantool's SQL does not. Authorization occurs via NoSQL requests.

The rules require that data must be physically independent (from underlying storage changes) and logically independent (from application program changes). So far there is not enough experience to make this guarantee.

The rules require certain types of updatable views. Tarantool's views are not updatable.

The rules state that it should be impossible to use a low-level language to bypass integrity as defined in the relational-level language. In Tarantool's case, this is not true, for example one can execute a request with Tarantool's NoSQL to violate a foreign-key constraint that was defined with Tarantool's SQL.

To learn more about SQL in Tarantool, check the *reference*.

### 3.9.2 SQL tutorial

This tutorial is a demonstration of the support for SQL in Tarantool. It includes the functionality that you'd encounter in an «SQL-101» course.

#### Prerequisites

Before starting this tutorial:

1. Install the *tt CLI* utility.
2. Create a tt environment in the current directory using the *tt init* command.
3. Inside the `instances.enabled` directory of the created tt environment, create the `sql_tutorial` directory.
4. Inside `instances.enabled/sql_tutorial`, create the `instances.yml` and `config.yaml` files:
   - `instances.yml` specifies instances to run in the current environment:

     ```
     instance001:
     ```

   - `config.yaml` contains basic *configuration*:

     ```
     groups:
       group001:
         replicasets:
           replicaset001:
             instances:
               instance001:
                 iproto:
                   listen:
                   - uri: '127.0.0.1:3301'
     ```

Read more: *Starting instances using the tt utility*.

### Create a table and execute SQL statements

#### Start Tarantool

After configuration, start a Tarantool instance from the tt environment directory using the following command:

```
$ tt start sql_tutorial
```

After that, connect to the instance:

```
$ tt connect sql_tutorial:instance001
```

This command opens an interactive Tarantool console. Now you can start working with the database.

#### Switch to the SQL language

A feature of the client console program is that you can switch languages and specify the end-of-statement delimiter.

Run the following commands to set the *console input* language to SQL and use semicolon as a delimiter:

```
sql_tutorial:instance001> \set language sql
sql_tutorial:instance001> \set delimiter ;
```

### CREATE, INSERT, UPDATE, SELECT

To get started, enter these SQL statements:

```
CREATE TABLE table1 (column1 INTEGER PRIMARY KEY, column2 VARCHAR(100));
INSERT INTO table1 VALUES (1, 'A');
UPDATE table1 SET column2 = 'B';
SELECT * FROM table1 WHERE column1 = 1;
```

The result of the `SELECT` statement looks like this:

```
sql_tutorial:instance001> SELECT * FROM table1 WHERE column1 = 1;
---
- metadata:
  - name: COLUMN1
    type: integer
  - name: COLUMN2
    type: string
  rows:
  - [1, 'B']
...
```

The result includes:

- metadata: the names and data types of each column
- result rows

For conciseness, metadata is skipped in query results in this tutorial. Only the result rows are shown.

### CREATE TABLE

Here is `CREATE TABLE` with more details:

- There are multiple columns, with different data types.
- There is a `PRIMARY KEY` (unique and not-null) for two of the columns.

Create another table:

```
CREATE TABLE table2 (column1 INTEGER,
                     column2 VARCHAR(100),
                     column3 SCALAR,
                     column4 DOUBLE,
                     PRIMARY KEY (column1, column2));
```

The result is: `row_count: 1`.

### INSERT

Put four rows in the table (`table2`):

- The INTEGER and DOUBLE columns get numbers
- The VARCHAR and SCALAR columns get strings (the SCALAR strings are expressed as hexadecimals)

```
INSERT INTO table2 VALUES (1, 'AB', X'4142', 5.5);
INSERT INTO table2 VALUES (1, 'CD', X'2020', 1E4);
INSERT INTO table2 VALUES (2, 'AB', X'2020', 12.34567);
INSERT INTO table2 VALUES (-1000, '', X'', 0.0);
```

Then try to put another row:

```
INSERT INTO table2 VALUES (1, 'AB', X'A5', -5.5);
```

This `INSERT` fails because of a primary-key violation: the row with the primary key `1, 'AB'` already exists.

#### The SEQSCAN keyword

Sequential scan is the scan through all the table rows instead of using indexes. In Tarantool, `SELECT` SQL queries that perform sequential scans are prohibited by default. For example, this query leads to the error `Scanning is not allowed for 'table2'`:

```
SELECT * FROM table2;
```

To execute a scan query, put the `SEQSCAN` keyword before the table name:

```
SELECT * FROM SEQSCAN table2;
```

Try to execute these queries that use indexed `column1` in filters:

```
SELECT * FROM table2 WHERE column1 = 1;
SELECT * FROM table2 WHERE column1 + 1 = 2;
```

The result is:

- The first query returns rows:

```
- [1, 'AB', 'AB', 10.5]
- [1, 'CD', '  ', 10005]
```

- The second query fails with the error `Scanning is not allowed for 'TABLE2'`. Although `column1` is indexed, the expression `column1 + 1` is not calculated from the index, which makes this `SELECT` a scan query.

---

**Примечание:** To enable SQL scan queries without `SEQSCAN` for the current session, run this command:

```
SET SESSION "sql_seq_scan" = true;
```

---

Learn more about using `SEQSCAN` in the *SQL FROM clause description*.

### SELECT with ORDER BY clause

Retrieve the 4 rows in the table, in descending order by `column2`, then (where the `column2` values are the same) in ascending order by `column4`.

`*` is short for «all columns».

```
SELECT * FROM SEQSCAN table2 ORDER BY column2 DESC, column4 ASC;
```

The result is:

```
- - [1, 'CD', '  ', 10000]
  - [1, 'AB', 'AB', 5.5]
  - [2, 'AB', '  ', 12.34567]
  - [-1000, '', '', 0]
```

### SELECT with WHERE clauses

Retrieve some of what you inserted:

- The first statement uses the `LIKE` comparison operator which is asking for «first character must be „A“, the next characters can be anything.»
- The second statement uses logical operators and parentheses, so the `AND` expressions must be true, or the `OR` expression must be true. Notice the columns don't have to be indexed.

```
SELECT column1, column2, column1 * column4 FROM SEQSCAN table2 WHERE column2
LIKE 'A%';
SELECT column1, column2, column3, column4 FROM SEQSCAN table2
    WHERE (column1 < 2 AND column4 < 10)
    OR column3 = X'2020';
```

The first result is:

```
- - [1, 'AB', 5.5]
  - [2, 'AB', 24.69134]
```

The second result is:

```
- - [-1000, '', '', 0]
  - [1, 'AB', 'AB', 5.5]
  - [1, 'CD', '  ', 10000]
  - [2, 'AB', '  ', 12.34567]
```

#### SELECT with GROUP BY and aggregate functions

Retrieve with grouping.

The rows that have the same values for `column2` are grouped and are aggregated – summed, counted, averaged – for `column4`.

```
SELECT column2, SUM(column4), COUNT(column4), AVG(column4)
FROM SEQSCAN table2
GROUP BY column2;
```

The result is:

```
- - ['', 0, 1, 0]
  - ['AB', 17.84567, 2, 8.922835]
  - ['CD', 10000, 1, 10000]
```

### Complications and complex SELECTs

#### NULLs

Insert rows that contain `NULL` values.

`NULL` is not the same as Lua `nil`; it commonly is used in SQL for unknown or not-applicable.

```
INSERT INTO table2 VALUES (1, NULL, X'4142', 5.5);
INSERT INTO table2 VALUES (0, '!!@', NULL, NULL);
INSERT INTO table2 VALUES (0, '!!!', X'00', NULL);
```

The results are:

- The first `INSERT` fails because `NULL` is not permitted for a column that was defined with a `PRIMARY KEY` clause.
- The other `INSERT` statements succeed.

#### Индексы

Create a new index on `column4`.

There already is an index for the primary key. Indexes are useful for making queries faster. In this case, the index also acts as a constraint, because it prevents two rows from having the same values in `column4`. However, it is not an error that `column4` has multiple occurrences of NULLs.

```
CREATE UNIQUE INDEX i ON table2 (column4);
```

The result is: `rowcount: 1`.

### Create a subset table

Create a table `table3`, which contains a subset of the `table2` columns and a subset of the `table2` rows.

You can do this by combining `INSERT` with `SELECT`. Then select everything from the result table.

```
CREATE TABLE table3 (column1 INTEGER, column2 VARCHAR(100), PRIMARY KEY
(column2));
INSERT INTO table3 SELECT column1, column2 FROM SEQSCAN table2 WHERE column1 <> 2;
SELECT * FROM SEQSCAN table3;
```

The result is:

```
- - [-1000, '']
  - [0, '!!!']
  - [0, '!!@']
  - [1, 'AB']
  - [1, 'CD']
```

### SELECT with a subquery

A subquery is a query within a query.

Find all the rows in `table2` whose `(column1, column2)` values are not present in `table3`.

```
SELECT * FROM SEQSCAN table2 WHERE (column1, column2) NOT IN (SELECT column1,
column2 FROM SEQSCAN table3);
```

The result is the single row that was excluded when inserting the rows with the `INSERT ... SELECT` statement:

```
- - [2, 'AB', '  ', 12.34567]
```

### SELECT with a join

A join is a combination of two tables. There is more than one way to do them in Tarantool, for example, «Cartesian joins» or «left outer joins».

This example shows the most typical case, where column values from one table match column values from another table.

```
SELECT * FROM SEQSCAN table2, table3
    WHERE table2.column1 = table3.column1 AND table2.column2 = table3.column2
    ORDER BY table2.column4;
```

The result is:

```
- - [0, '!!!', "\0", null, 0, '!!!']
  - [0, '!!@', null, null, 0, '!!@']
  - [-1000, '', '', 0, -1000, '']
  - [1, 'AB', 'AB', 5.5, 1, 'AB']
  - [1, 'CD', ' ', 10000, 1, 'CD']
```

### Constraints and foreign keys

#### CREATE TABLE with a CHECK clause

Create a table that includes a constraint – there must not be any rows containing 13 in column2. After that, try to insert the following row:

```
CREATE TABLE table4 (column1 INTEGER PRIMARY KEY, column2 INTEGER, CHECK
(column2 <> 13));
INSERT INTO table4 VALUES (12, 13);
```

Result: the insert fails, as it should, with the message `Check constraint 'ck_unnamed_TABLE4_1' failed for tuple`.

#### CREATE TABLE with a FOREIGN KEY clause

Create a table that includes a constraint: there must not be any rows containing values that do not appear in table2.

```
CREATE TABLE table5 (column1 INTEGER, column2 VARCHAR(100),
    PRIMARY KEY (column1),
    FOREIGN KEY (column1, column2) REFERENCES table2 (column1, column2));
INSERT INTO table5 VALUES (2,'AB');
INSERT INTO table5 VALUES (3,'AB');
```

Result:

- The first INSERT statement succeeds because table3 contains a row with `[2, 'AB', ' ', 12.34567]`.
- The second INSERT statement, correctly, fails with the message `Foreign key constraint ''fk_unnamed_TABLE5_1'' failed: foreign tuple was not found`.

#### UPDATE

Due to earlier INSERT statements, these values are in column4 of table2: {0, NULL, NULL, 5.5, 10000, 12.34567}. Add 5 to each of these values except 0. Adding 5 to NULL results in NULL, as SQL arithmetic requires. Use SELECT to see what happened to column4.

```
UPDATE table2 SET column4 = column4 + 5 WHERE column4 <> 0;
SELECT column4 FROM SEQSCAN table2 ORDER BY column4;
```

The result is: {NULL, NULL, 0, 10.5, 17.34567, 10005}.

### DELETE

Due to earlier `INSERT` statements, there are 6 rows in `table2`:

```
- - [-1000, '', '', 0]
  - [0, '!!!', "\0", null]
  - [0, '!!@', null, null]
  - [1, 'AB', 'AB', 10.5]
  - [1, 'CD', '  ', 10005]
  - [2, 'AB', '  ', 17.34567]
```

Try to delete the last and first of these rows:

```
DELETE FROM table2 WHERE column1 = 2;
DELETE FROM table2 WHERE column1 = -1000;
SELECT COUNT(column1) FROM SEQSCAN table2;
```

The result is:

- The first `DELETE` statement causes an error because there's a foreign-key constraint.
- The second `DELETE` statement succeeds.
- The `SELECT` statement shows that there are 5 rows remaining.

### ALTER TABLE with a FOREIGN KEY clause

Create another constraint that there must not be any rows in `table1` containing values that do not appear in `table5`. This was impossible during the `table1` creation because at that time `table5` did not exist. You can add constraints to existing tables with the `ALTER TABLE` statement.

```
ALTER TABLE table1 ADD CONSTRAINT c
    FOREIGN KEY (column1) REFERENCES table5 (column1);
DELETE FROM table1;
ALTER TABLE table1 ADD CONSTRAINT c
    FOREIGN KEY (column1) REFERENCES table5 (column1);
```

Result: the `ALTER TABLE` statement fails the first time because there is a row in `table1`, and `ADD CONSTRAINT` requires that the table be empty. After the row is deleted, the `ALTER TABLE` statement completes successfully. Now there is a chain of references, from `table1` to `table5` and from `table5` to `table2`.

### Триггеры

The idea of a trigger is: if a change (`INSERT` or `UPDATE` or `DELETE`) happens, then a further action – perhaps another `INSERT` or `UPDATE` or `DELETE` – will happen.

Set up the following trigger: when a update to `table3` is done, do an update to `table2`. Specify this as `FOR EACH ROW`, so that the trigger activates 5 times (since there are 5 rows in `table3`).

```
SELECT column4 FROM table2 WHERE column1 = 2;
CREATE TRIGGER tr AFTER UPDATE ON table3 FOR EACH ROW
BEGIN UPDATE table2 SET column4 = column4 + 1 WHERE column1 = 2; END;
UPDATE table3 SET column2 = column2;
SELECT column4 FROM table2 WHERE column1 = 2;
```

Result:

- The first `SELECT` shows that the original value of `column4` in `table2` where `column1 = 2` was: 17.34567.
- The second `SELECT` returns:

```
- - [22.34567]
```

### Operators and functions

#### String operations

You can manipulate string data (usually defined with `CHAR` or `VARCHAR` data types) in many ways. For example:

- concatenate strings with the `||` operator
- extract substrings with the `SUBSTR` function

```
SELECT column2, column2 || column2, SUBSTR(column2, 2, 1) FROM SEQSCAN table2;
```

The result is:

```
- - ['!!!', '!!!!!!!', '!']
  - ['!!@', '!!@!!@', '!']
  - ['AB', 'ABAB', 'B']
  - ['CD', 'CDCD', 'D']
  - ['AB', 'ABAB', 'B']
```

#### Number operations

You can also manipulate number data (usually defined with `INTEGER` or `DOUBLE` data types) in many ways. For example:

- shift left with the `<<` operator
- get modulo with the `%` operator

```
SELECT column1, column1 << 1, column1 << 2, column1 % 2 FROM SEQSCAN table2;
```

The result is:

```
- - [0, 0, 0, 0]
  - [0, 0, 0, 0]
  - [1, 2, 4, 1]
  - [1, 2, 4, 1]
  - [2, 4, 8, 0]
```

### Ranges and limits

Tarantool can handle:

- integers anywhere in the 4-byte integer range
- approximate-numerics anywhere in the 8-byte IEEE floating point range
- any Unicode characters, with UTF-8 encoding and a choice of collations

Insert such values in a new table and see what happens when you select them with arithmetic on a number column and ordering by a string column.

```
CREATE TABLE t6 (column1 INTEGER, column2 VARCHAR(10), column4 DOUBLE,
PRIMARY KEY (column1));
INSERT INTO t6 VALUES (-1234567890, 'АБВГД', 123456.123456);
INSERT INTO t6 VALUES (+1234567890, 'GD', 1e30);
INSERT INTO t6 VALUES (10, 'FADEW?', 0.000001);
INSERT INTO t6 VALUES (5, 'ABCDEFG', NULL);
SELECT column1 + 1, column2, column4 * 2 FROM SEQSCAN t6 ORDER BY column2;
```

The result is:

```
- - [6, 'ABCDEFG', null]
  - [11, 'FADEW?', 2e-06]
  - [1234567891, 'GD', 2e+30]
  - [-1234567889, 'АБВГД', 246912.246912]
```

### Views

A view (or *viewed table*), is virtual, meaning that its rows aren't physically in the database, their values are calculated from other tables.

Create a view `v3` based on `table3` and select from it:

```
CREATE VIEW v3 AS SELECT SUBSTR(column2,1,2), column4 FROM SEQSCAN t6
WHERE column4 >= 0;
SELECT * FROM v3;
```

The result is:

```
- - ['АБ', 123456.123456]
  - ['FA', 1e-06]
  - ['GD', 1e+30]
```

### Common table expressions

By putting `WITH` + `SELECT` in front of a `SELECT`, you can make a temporary view that lasts for the duration of the statement.

Create such a view and select from it:

```
WITH cte AS (
             SELECT SUBSTR(column2,1,2), column4 FROM SEQSCAN t6
             WHERE column4 >= 0)
SELECT * FROM cte;
```

The result is the same as the `CREATE VIEW` result:

```
- - ['АБ', 123456.123456]
  - ['FA', 1e-06]
  - ['GD', 1e+30]
```

## VALUES

Tarantool can handle statements like `SELECT 55;` (select without `FROM`) like some other popular DBMSs. But it also handles the more standard statement `VALUES (expression [, expression ...]);`.

```
SELECT 55 * 55, 'The rain in Spain';
VALUES (55 * 55, 'The rain in Spain');
```

The result of both these statements is:

```
- - [3025, 'The rain in Spain']
```

### Metadata

To find out the internal structure of the Tarantool database with SQL, select from the Tarantool system tables `_space`, `_index`, and `_trigger`:

```
SELECT * FROM SEQSCAN "_space";
SELECT * FROM SEQSCAN "_index";
SELECT * FROM SEQSCAN "_trigger";
```

Actually, these statements select from NoSQL «system spaces».

Select from `_space` by a table name:

```
SELECT "id", "name", "owner", "engine" FROM "_space" WHERE "name"='TABLE3';
```

The result is:

```
- - [517, 'TABLE3', 1, 'memtx']
```

### Using SQL from Lua

You can execute SQL statements directly from the Lua code without switching to the SQL input.

Change the settings so that the console accepts statements written in Lua instead of statements written in SQL:

```
sql_tutorial:instance001> \set language lua
```

### box.execute()

You can invoke SQL statements using the Lua function `box.execute(string)`.

```
sql_tutorial:instance001> box.execute([[SELECT * FROM SEQSCAN table3;]]);
```

The result is:

```
- - [-1000, '']
  - [0, '!!!']
  - [0, '!!@']
  - [1, 'AB']
  - [1, 'CD']
...
```

### Create a million-row table

To see how the SQL in Tarantool scales, create a bigger table.

The following Lua code generates one million rows with random data and inserts them into a table. Copy this code into the Tarantool console and wait a bit:

```
box.execute("CREATE TABLE tester (s1 INT PRIMARY KEY, s2 VARCHAR(10))");

function string_function()
    local random_number
    local random_string
    random_string = ""
    for x = 1, 10, 1 do
        random_number = math.random(65, 90)
        random_string = random_string .. string.char(random_number)
    end
    return random_string
end;

function main_function()
    local string_value, t, sql_statement
    for i = 1, 1000000, 1 do
        string_value = string_function()
        sql_statement = "INSERT INTO tester VALUES (" .. i .. ",'" .. string_value .. "')
↪"
        box.execute(sql_statement)
    end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
end;
start_time = os.clock();
main_function();
end_time = os.clock();
print('insert done in ' .. end_time - start_time .. ' seconds');
```

The result is: you now have a table with a million rows, with a message saying `insert done in 88.570578 seconds`.

### Select from a million-row table

Check how `SELECT` works on the million-row table:

- the first query goes by an index because `s1` is the primary key
- the second query does not go by an index

```
box.execute([[SELECT * FROM tester WHERE s1 = 73446;]]);
box.execute([[SELECT * FROM SEQSCAN tester WHERE s2 LIKE 'QFML%';]]);
```

The result is:

- the first statement completes instantaneously
- the second statement completed noticeably slower

### Cleanup and exit

To cleanup all the objects created in this tutorial, switch to the SQL input language again. Then run the `DROP` statements for all created tables, views, and triggers.

These statements must be entered separately.

```
sql_tutorial:instance001> \set language sql
sql_tutorial:instance001> DROP TABLE tester;
sql_tutorial:instance001> DROP TABLE table1;
sql_tutorial:instance001> DROP VIEW v3;
sql_tutorial:instance001> DROP TRIGGER tr;
sql_tutorial:instance001> DROP TABLE table5;
sql_tutorial:instance001> DROP TABLE table4;
sql_tutorial:instance001> DROP TABLE table3;
sql_tutorial:instance001> DROP TABLE table2;
sql_tutorial:instance001> DROP TABLE t6;
sql_tutorial:instance001> \set language lua
sql_tutorial:instance001> os.exit();
```

### 3.9.3 Улучшаем работу MySQL с помощью Tarantool

Replicating MySQL is one of the Tarantool's killer functions. It allows you to keep your existing MySQL database while at the same time accelerating it and scaling it out horizontally. Even if you aren't interested in extensive expansion, replacing existing replicas with Tarantool can save you money, because Tarantool is more efficient per core than MySQL. To read a testimonial of a company that implemented Tarantool replication on a large scale, see the following article.

If you run into any trouble with regards to the basics of Tarantool, see the *Getting started guide* or the *Data model description*. A helpful log for troubleshooting during this tutorial is `replicatord.log` in `/var/log`. You can also have a look at the instance's log `example.log` in `/var/log/tarantool`.

The tutorial is intended for **CentOS 7.5** and **MySQL 5.7**. The tutorial requires that `systemd` and MySQL are installed.

#### Setting up MySQL

In this section, you configure MySQL and create a database.

1. First, install the necessary packages in CentOS:

```
$ yum -y install git ncurses-devel cmake gcc-c++ boost boost-devel wget unzip nano␣
↪bzip2 mysql-devel mysql-lib
```

2. Clone the Tarantool-MySQL replication package from GitHub:

```
$ git clone https://github.com/tarantool/mysql-tarantool-replication.git
```

3. Build the replicator with `cmake`:

```
$ cd mysql-tarantool-replication
$ git submodule update --init --recursive
$ cmake .
$ make
```

4. The replicator will run as a `systemd` daemon called `replicatord`, so, edit its `systemd` service file (`replicatord.service`) in the `mysql-tarantool-replication` repository:

```
$ nano replicatord.service
```

   The following line should be changed:

```
ExecStart=/usr/local/sbin/replicatord -c /usr/local/etc/replicatord.cfg
```

   To change it, replace the `.cfg` extension with `.yml`:

```
ExecStart=/usr/local/sbin/replicatord -c /usr/local/etc/replicatord.yml
```

5. Next, copy the files from the `replicatord` repository to other necessary locations:

```
$ cp replicatord /usr/local/sbin/replicatord
$ cp replicatord.service /etc/systemd/system
```

6. Enter MySQL console and create a sample database (depending on your existing installation, you may be a user other than root):

```
mysql -u root -p
CREATE DATABASE menagerie;
QUIT
```

7. Get some sample data from MySQL. The data will be pulled into the root directory. After that, install it from the terminal.

```
cd
wget http://downloads.mysql.com/docs/menagerie-db.zip
unzip menagerie-db.zip
cd menagerie-db
mysql -u root -p menagerie < cr_pet_tbl.sql
mysql -u root -p menagerie < load_pet_tbl.sql
mysql menagerie -u root -p < ins_puff_rec.sql
mysql menagerie -u root -p < cr_event_tbl.sql
```

8. Enter MySQL console and massage the data for use with the Tarantool replicator. In this step, you:
   - add an ID
   - change a field name to avoid conflict
   - cut down the number of fields

   With real data, this is the step that involves the most tweaking.

```
mysql -u root -p
USE menagerie;
ALTER TABLE pet ADD id INT PRIMARY KEY AUTO_INCREMENT FIRST;
ALTER TABLE pet CHANGE COLUMN 'name' 'name2' VARCHAR(255);
ALTER TABLE pet DROP sex, DROP birth, DROP death;
QUIT
```

9. The sample data is set up. Edit MySQL configuration file to use it with the replicator:

```
$ cd
$ nano /etc/my.cnf
```

   Обратите внимание, что ваш `my.cnf` для MySQL может находиться в другом месте. Задайте:

```
[mysqld]
binlog_format = ROW
server_id = 1
log-bin = mysql-bin
interactive_timeout = 3600
wait_timeout = 3600
max_allowed_packet = 32M
socket = /var/lib/mysql/mysql.sock
bind-address = 127.0.0.1

[client]
socket = /var/lib/mysql/mysql.sock
```

10. After exiting `nano`, restart `mysqld`:

```
$ systemctl restart mysqld
```

### Installing and configuring Tarantool

In this section, you install Tarantool and set up spaces for replication.

1. Go to the Download page and follow the installation instructions.
2. Install the *tt CLI* utility.
3. Create a new tt environment in the current directory using the *tt init* command.
4. In the `/etc/tarantool/instances.available/mysql` directory, create the `tt` instance configuration files:

   - `config.yaml` – specifies the following configuration

   ```
   app:
     file: 'myapp.lua'

   groups:
     group001:
       replicasets:
         replicaset001:
           instances:
             instance001:
               iproto:
                 listen:
                 - uri: '127.0.0.1:3301'
   ```

   - `instances.yml` – specifies instances to run in the current environment

   ```
   instance001:
   ```

   - `myapp.lua` – contains a Lua script with an application to load

   ```
   box.schema.user.grant('guest', 'read,write,execute', 'universe')

   local function bootstrap()
       if not box.space.mysqldaemon then
           s = box.schema.space.create('mysqldaemon')
           s:create_index('primary',
                   { type = 'tree', parts = { 1, 'unsigned' }, if_not_exists =␣
   ↪true })
       end
       if not box.space.mysqldata then
           t = box.schema.space.create('mysqldata')
           t:create_index('primary',
                   { type = 'tree', parts = { 1, 'unsigned' }, if_not_exists =␣
   ↪true })
       end
   end
   bootstrap()
   ```

   For details, see the *Configuration* section.

5. Inside the `instances.enabled` directory of the created tt environment, create a symlink (`mysql`) to the directory from the previous step:

```
$ ln -s /etc/tarantool/instances.available/mysql mysql
```

6. Next, start up the Lua program with `tt`, the Tarantool command-line utility:

```
$ tt start mysql
```

7. Enter the Tarantool instance:

```
$ tt connect mysql:instance001
```

8. Check that the target spaces were successfully created:

```
mysql:instance001> box.space._space:select()
```

   At the bottom, you will see `mysqldaemon` and `mysqldata` spaces. Then exit with «CTRL+C».

### Setting up the replicator

MySQL and Tarantool are now set up. You can proceed to configure the replicator.

1. Edit the `replicatord.yml` file in the main `tarantool-mysql-replication` directory:

```
nano replicatord.yml
```

2. Change the entire file as follows. Don't forget to add your MySQL password and set the appropriate user:

```
mysql:
    host: 127.0.0.1
    port: 3306
    user: root
    password:
    connect_retry: 15 # seconds

tarantool:
    host: 127.0.0.1:3301
    binlog_pos_space: 512
    binlog_pos_key: 0
    connect_retry: 15 # seconds
    sync_retry: 1000 # milliseconds

mappings:
 - database: menagerie
   table: pet
   columns: [ id, name2, owner, species ]
   space: 513
   key_fields:  [ 0 ]
   # insert_call: function_name
   # update_call: function_name
   # delete_call: function_name
```

3. Copy `replicatord.yml` to the location where `systemd` looks for it:

```
$ cp replicatord.yml /usr/local/etc/replicatord.yml
```

4. Next, start up the replicator:

```
$ systemctl start replicatord
```

5. Enter the Tarantool instance:

```
$ tt connect mysql:instance001
```

6. Do a select on the `mysqldata` space. The replicated content from MySQL looks the following way:

```
mysql:instance001> box.space.mysqldata:select()
---
- - [1, 'Fluffy', 'Harold', 'cat']
  - [2, 'Claws', 'Gwen', 'cat']
  - [3, 'Buffy', 'Harold', 'dog']
  - [4, 'Fang', 'Benny', 'dog']
  - [5, 'Bowser', 'Diane', 'dog']
  - [6, 'Chirpy', 'Gwen', 'bird']
  - [7, 'Whistler', 'Gwen', 'bird']
  - [8, 'Slim', 'Benny', 'snake']
  - [9, 'Puffball', 'Diane', 'hamster']
```

### Testing the replication

In this section, you enter a record into MySQL and check that the record is replicated to Tarantool. To do this:

1. Exit the Tarantool instance with `CTRL-D`.
2. Insert a record into MySQL:

```
mysql -u root -p
USE menagerie;
INSERT INTO pet(name2, owner, species) VALUES ('Spot', 'Brad', 'dog');
QUIT
```

3. In the terminal, enter the Tarantool instance:

```
$ tt connect mysql:instance001
```

4. To see the replicated data in Tarantool, run the following command:

```
mysql:instance001> box.space.mysqldata:select()
```

## 3.10 More guides

### 3.10.1 Understanding the binary protocol

#### Overview

To communicate with each other, Tarantool instances use a binary protocol called iproto. To learn more, see the *Binary protocol* section.

In this set of examples, the user will be looking at binary code transferred via iproto. The code is intercepted with `tcpdump`, a monitoring utility.

#### Примеры

Чтобы выполнить примеры, приведенные в этом разделе, запустите на компьютере с Linux три командных оболочки (терминала).

– На терминале №1 запустите мониторинг порта 3302 с помощью tcpdump:

```
sudo tcpdump -i lo 'port 3302' -X
```

На терминале №2 запустите сервер так:

```
box.cfg{listen=3302}
box.schema.space.create('tspace')
box.space.tspace:create_index('I')
box.space.tspace:insert{280}
box.schema.user.grant('guest','read,write,execute,create,drop','universe')
```

На терминале №3 запустите ещё один сервер, который будет выступать в качестве клиента, так:

```
box.cfg{}
net_box = require('net.box')
conn = net_box.connect('localhost:3302')
```

#### IPROTO_SELECT

On terminal #3, run the following:

```
conn.space.tspace:select(280)
```

Теперь посмотрите, что tcpdump покажет для запроса на подключение к порту 3302. После слов «length 32» идет пакет, который заканчивается этими 32 байтами (комментарии отделены отступами):

```
ce 00 00 00 1b   MP_UINT = 27, десятичное число = число байт после этого
82               MP_MAP, размер 2 (назовем это Main-Map)
01                 IPROTO_SYNC (1-й элемент Main-Map)
04                 MP_INT = 4 = число, которое увеличивается на 1 с каждым запросом
00                 IPROTO_REQUEST_TYPE (2-й элемент Main-Map)
01                 IPROTO_SELECT
86                 MP_MAP, размер 6 (назовем это Select-Map)
10                   IPROTO_SPACE_ID (1-й элемент Select-Map)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
cd 02 00             MP_UINT = 512, десятичное число = id tspace (может быть больше)
11                   IPROTO_INDEX_ID (2-й элемент Select-Map)
00                   MP_INT = 0 = id индекса в tspace
14                   IPROTO_ITERATOR (3-й элемент Select-Map)
00                   MP_INT = 0 = константа Tarantool iterator_type.h ITER_EQ
13                   IPROTO_OFFSET (4-й элемент Select-Map)
00                   MP_INT = 0 = смещение
12                   IPROTO_LIMIT (5-й элемент Select-Map)
ce ff ff ff ff       MP_UINT = 4294967295 = наибольший возможный предел
20                   IPROTO_KEY (6-й элемент Select-Map)
91                   MP_ARRAY, размер 1 (назовем это Key-Array)
cd 01 18               MP_UINT = 280 (6-й элемент Select-Map, 1-й элемент Key-Array)
                       -- 280, ключевое значение, которое мы ищем
```

Теперь в файле исходного кода net_box.c перейдите к строке `netbox_encode_select(lua_State *L)`. Из комментариев и из простых вызовов функций типа `mpstream_encode_uint(&stream, IPROTO_SPACE_ID)`; можно понять, как net_box собирает воедино содержимое пакета, описанного выше с помощью tcpdump.

Существуют библиотеки для чтения и записи объектов в формате MessagePack. Программисты на языке C иногда включают msgpuck.h.

Теперь вы знаете, как сам Tarantool выполняет запросы по бинарному протоколу. Если какие-то детали остаются неясными, обратитесь к файлу `net_box.c`, где описаны процедуры для каждого запроса. Некоторые *коннекторы* написаны аналогично.

### IPROTO_UPDATE

Рассмотрим пример IPROTO_UPDATE. Предположим, пользователь изменяет поле №2 кортежа №2 в спейсе №256 на `'BBBB'`. Тело будет выглядеть так (обратите внимание, что в этом случае дополнительный необязательный элемент ассоциативного массива IPROTO_INDEX_BASE подчеркивает, что номера полей начинаются с 1 — это можно опустить):

```
04                 IPROTO_UPDATE
85                 IPROTO_MAP, размер 5
10                   IPROTO_SPACE_ID, 1-й элемент ассоциативного массива
cd 02 00             MP_UINT 256
11                   IPROTO_INDEX_ID, 2-й элемент ассоциативного массива
00                   MP_INT 0 = номер индекса первичного ключа
15                   IPROTO_INDEX_BASE, 3-й элемент ассоциативного массива
01                   MP_INT = 1, т.е. нумерация полей начинается с 1
21                   IPROTO_TUPLE, 4-й элемент ассоциативного массива
91                   MP_ARRAY, размер 1, для массива операций
93                     MP_ARRAY, размер 3
a1 3d                     MP_STR = OPERATOR = '='
02                        MP_INT = FIELD_NO = 2
a5 42 42 42 42 42         MP_STR = VALUE = 'BBBB'
20                   IPROTO_KEY, 5--й элемент ассоциативного массива
91                   MP_ARRAY, размер 1, для массива ключей
02                     MP_UINT = значение первичного ключа = 2
```

### IPROTO_EXECUTE

Пример байт-кода *IPROTO_EXECUTE*:

```
0b               IPROTO_EXECUTE
83               MP_MAP, размер 3
43                 IPROTO_STMT_ID 1-й элемент ассоциативного массива
ce d7 aa 74 1b     MP_UINT значение n.stmt_id
41                 IPROTO_SQL_BIND 2-й элемент ассоциативного массива
92                 MP_ARRAY, размер 2
01                   MP_INT = 1 = значение первого параметра
a1 61                MP_STR = 'a' = значение второго параметра
2b                 IPROTO_OPTIONS 3-й элемент ассоциативного массива
90                 MP_ARRAY, размер 0 (никакие опции не выбраны)
```

### IPROTO_INSERT

Пример байт-кода ответа на запрос box.space.space-name:insert{6}:

```
ce 00 00 00 20                MP_UINT = HEADER AND BODY SIZE
83                            MP_MAP, size 3
00                              IPROTO_REQUEST_TYPE
ce 00 00 00 00                  MP_UINT = IPROTO_OK
01                              IPROTO_SYNC
cf 00 00 00 00 00 00 00 53      MP_UINT = sync value
05                              IPROTO_SCHEMA_VERSION
ce 00 00 00 68                  MP_UINT = schema version
81                            MP_MAP, size 1
30                              IPROTO_DATA
dd 00 00 00 01                  MP_ARRAY, size 1 (row count)
91                              MP_ARRAY, size 1 (field count)
06                              MP_INT = 6 = the value that was inserted
```

### IPROTO_EVAL

Пример байт-кода ответа на запрос `conn:eval([[box.schema.space.create('_space');]])`:

```
ce 00 00 00 3b                   MP_UINT = HEADER AND BODY SIZE
83                               MP_MAP, size 3 (i.e. 3 items in header)
   00                               IPROTO_REQUEST_TYPE
   ce 00 00 80 0a                   MP_UINT = hexadecimal 800a
   01                               IPROTO_SYNC
   cf 00 00 00 00 00 00 00 26       MP_UINT = sync value
   05                               IPROTO_SCHEMA_VERSION
   ce 00 00 00 78                   MP_UINT = schema version value
   81                               MP_MAP, size 1
     31                               IPROTO_ERROR_24
     db 00 00 00 1d 53 70 61 63 etc. MP_STR = "Space '_space' already exists"
```

### Creating a table with IPROTO_EXECUTE

Byte codes, if we use the same net.box connection that we used in the beginning and we say `conn:execute([[CREATE TABLE t1 (dd INT PRIMARY KEY AUTOINCREMENT, дд STRING COLLATE "unicode");]])` `conn:execute([[INSERT INTO t1 VALUES (NULL, 'a'), (NULL, 'b');]])` and we watch what tcpdump displays, we will see two noticeable things: (1) the CREATE statement caused a schema change so the response has a new IPROTO_SCHEMA_VERSION value and the body includes the new contents of some system tables (caused by requests from net.box which users will not see); (2) the final bytes of the response to the INSERT will be:

```
81   MP_MAP, размер 1
42     IPROTO_SQL_INFO
82     MP_MAP, размер 2
00       константа Tarantool (не из iproto_constants.h) = SQL_INFO_ROW_COUNT
02       1 = число строк
01       константа Tarantool (не из iproto_constants.h) = SQL_INFO_AUTOINCREMENT_ID
92       MP_ARRAY, размер 2
01         первое число с автоинкрементом
02         второе число с автоинкрементом
```

### SELECT with SQL

Пример байт-кода SQL SELECT. Запросим полные метаданные, вызвав `conn.space._session_settings:update('sql_full_metadata', {{'=', 'value', true}})`, и выберем две строки из только что созданной таблицы: `conn:execute([[SELECT dd, дд AS д FROM t1;]])`. tcpdump выдаст следующий ответ (после заголовка):

```
82                      MP_MAP, размер 2 (метаданные и строки)
32                        IPROTO_METADATA
92                        MP_ARRAY, размер 2 (2 столбца)
85                          MP_MAP, размер 5 (5 элементов для столбца 1)
00 a2 44 44                   IPROTO_FIELD_NAME и 'DD'
01 a7 69 6e 74 65 67 65 72    IPROTO_FIELD_TYPE и 'integer'
03 c2                         IPROTO_FIELD_IS_NULLABLE и false
04 c3                         IPROTO_FIELD_IS_AUTOINCREMENT и true
05 c0                         PROTO_FIELD_SPAN и nil
85                          MP_MAP, размер 5 (5 элементов для столбца 2)
00 a2 d0 94                   IPROTO_FIELD_NAME и 'Д' в верхнем регистре
01 a6 73 74 72 69 6e 67       IPROTO_FIELD_TYPE и 'string'
02 a7 75 6e 69 63 6f 64 65    IPROTO_FIELD_COLL и 'unicode'
03 c3                         IPROTO_FIELD_IS_NULLABLE и true
05 a4 d0 b4 d0 b4             IPROTO_FIELD_SPAN и 'дд' в нижнем регистре
30                        IPROTO_DATA
92                        MP_ARRAY, размер 2
92                          MP_ARRAY, размер 2
01                            MP_INT = 1: содержимое строки 1, столбца 1
a1 61                         MP_STR = 'a': содержимое строки 1, столбца 2
92                          MP_ARRAY, размер 2
02                            MP_INT = 2: содержимое строки 2, столбца 1
a1 62                         MP_STR = 'b': содержимое строки 2, столбца 2
```

### IPROTO_PREPARE

Пример байт-кода SQL PREPARE. Если вызвать `conn:prepare([[SELECT dd, дд AS д FROM t1;]])`, вывод tcpdump будет почти таким же, но исчезнет IPROTO_DATA. Вместо этого появятся дополнительные байты:

```
34                      IPROTO_BIND_COUNT
00                      MP_UINT = 0

33                      IPROTO_BIND_METADATA
90                      MP_ARRAY, размер 0
```

`MP_UINT = 0`. Массив `MP_ARRAY` имеет размер 0, поскольку параметров нет. Вывод целиком:

```
84                      MP_MAP, размер 4
43                        IPROTO_STMT_ID
ce c2 3c 2c 1e            MP_UINT = ID инструкции
34                        IPROTO_BIND_COUNT
00                        MP_INT = 0 = число привязываемых параметров
33                        IPROTO_BIND_METADATA
90                        MP_ARRAY, размер 0 = нет привязываемых параметров
32                        IPROTO_METADATA
92                        MP_ARRAY, размер 2 (2 столбца)
85                          MP_MAP, размер 5 (5 элементов для столбца 1)
00 a2 44 44                   IPROTO_FIELD_NAME и 'DD'
01 a7 69 6e 74 65 67 65 72    IPROTO_FIELD_TYPE и 'integer'
03 c2                         IPROTO_FIELD_IS_NULLABLE и false
04 c3                         IPROTO_FIELD_IS_AUTOINCREMENT и true
05 c0                         PROTO_FIELD_SPAN и nil
85                          MP_MAP, размер 5 (5 элементов для столбца 2)
00 a2 d0 94                   IPROTO_FIELD_NAME + 'Д' в верхнем регистре
01 a6 73 74 72 69 6e 67       IPROTO_FIELD_TYPE и 'string'
02 a7 75 6e 69 63 6f 64 65    IPROTO_FIELD_COLL и 'unicode'
03 c3                         IPROTO_FIELD_IS_NULLABLE и true
05 a4 d0 b4 d0 b4             IPROTO_FIELD_SPAN и 'дд' в нижнем регистре
```

### Heartbeat

Пример байт-кода *контрольного сигнала*. Мастер может отправить следующее тело:

```
83                    MP_MAP, size 3
00                      Main-Map Item #1 IPROTO_REQUEST_TYPE
00                        MP_UINT = 0
02                      Main-Map Item #2 IPROTO_REPLICA_ID
02                        MP_UINT = 2 = id
04                      Main-Map Item #3 IPROTO_TIMESTAMP
cb                        MP_DOUBLE (MessagePack "Float 64")
41 d7 ba 06 7b 3a 03 21   8-byte timestamp
81                    MP_MAP (body), size 1
5a                    Body Map Item #1 IPROTO_VCLOCK_SYNC
14                      MP_UINT = 20 (vclock sync value)
```

Byte code for the *heartbeat* example. The replica might send back this body:

```
81                          MP_MAP, size 1
00                            Main-Map Item #1 IPROTO_REQUEST_TYPE
00                            MP_UINT = 0 = IPROTO_OK
83                            MP_MAP (body), size 3
26                              Body Map Item #1 IPROTO_VCLOCK
81                                MP_MAP, size 1 (vclock of 1 component)
01                                  MP_UINT = 1 = id (part 1 of vclock)
06                                  MP_UINT = 6 = lsn (part 2 of vclock)
5a                              Body Map Item #2 IPROTO_VCLOCK_SYNC
14                                MP_UINT = 20 (vclock sync value)
53                              Body Map Item #3 IPROTO_TERM
31                                MP_UINT = 49 (term value)
```

### 3.10.2 Практические задания по libslave

`libslave` представляет собой библиотеку C++ для считывания изменений данных, внесенных с помощью MySQL, а также – опционально – для записи их в базу данных Tarantool. Она выступает в качестве ведомого в схеме репликации. Сервер MySQL записывает информацию об изменении данных в бинарный журнал и передает ее на любой клиент, который запрашивает: «Хочу увидеть всю информацию, начиная с этого файла и этой записи, безостановочно». Таким образом, библиотека `libslave`, прежде всего, используется для создания реплик базы данных Tarantool (намного быстрее, чем используя традиционный ведомый сервер MySQL) и для отслеживания изменений данных, чтобы они были пригодны для поиска.

Здесь мы не будем подробно рассматривать библиотеку – информация есть в документации по API. Мы лишь дадим упражнение: минимальная программа с использованием библиотеки.

---

**Примечание:** Используйте тестовый сервер. Не используйте боевой сервер.

---

ШАГ 1: Убедитесь в наличии следующего:

- последняя версия Linux (например, Ubuntu версии 14.04 не подойдет),
- сервер MySQL версии 5.6 или 5.7 (MariaDB не подойдет),
- пакет программ для разработки клиента MySQL. Например, на Ubuntu можно загрузить его с помощью следующей команды:

```
$ sudo apt-get install mysql-client-core-5.7
```

ШАГ 2: Установите `libslave`.

Рекомендуется источник по ссылке https://github.com/tarantool/libslave/. Загрузки включают в себя только исходный код.

```
$ sudo apt-get install libboost-all-dev
$ cd ~
$ git clone https://github.com/tarantool/libslave.git tarantool-libslave
$ cd tarantool-libslave
$ git submodule init
$ git submodule update
$ cmake .
$ make
```

Если система выдаст сообщение с ошибкой со словом «vector», отредактируйте `field.h`, добавив следующую строку:

```
#include <vector>
```

ШАГ 3: Запустите сервер MySQL. В командной строке добавьте соответствующие коммутаторы для выполнения репликации. Например:

```
$ mysqld --log-bin=mysql-bin --server-id=1
```

ШАГ 4: Для целей данного упражнения, предполагаем, что у вас есть:

- пользователь «root» с паролем «root» с правами,
- тестовая база данных «test» с тестовой таблицей под названием «test»,
- бинарный журнал под названием «mysql-bin»,
- сервер с идентификатором 1.

Значения заданы в программе, хотя программу, конечно, можно изменить – посмотреть настройки несложно.

ШАГ 5: Обратите внимание на программу:

```
#include <unistd.h>
#include <iostream>
#include <sstream>
#include "Slave.h"
#include "DefaultExtState.h"

slave::Slave* sl = NULL;

void callback(const slave::RecordSet& event) {
    slave::Position sBinlogPos = sl->getLastBinlogPos();
    switch (event.type_event) {
    case slave::RecordSet::Update: std::cout << "UPDATE" << "\n"; break;
    case slave::RecordSet::Delete: std::cout << "DELETE" << "\n"; break;
    case slave::RecordSet::Write:  std::cout << "INSERT" << "\n"; break;
    default: break;
    }
}

bool isStopping()
{
    return 0;
}

int main(int argc, char** argv)
{
    slave::MasterInfo masterinfo;
    slave::Position position("mysql-bin", 0);
    masterinfo.conn_options.mysql_host = "127.0.0.1";
    masterinfo.conn_options.mysql_port = 3306;
    masterinfo.conn_options.mysql_user = "root";
    masterinfo.conn_options.mysql_pass = "root";
    bool error = false;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    try {
        slave::DefaultExtState sDefExtState;
        slave::Slave slave(masterinfo, sDefExtState);
        sl = &slave;
        sDefExtState.setMasterPosition(position);
        slave.setCallback("test", "test", callback);
        slave.init();
        slave.createDatabaseStructure();
        try {
            slave.get_remote_binlog(isStopping);
        } catch (std::exception& ex) {
            std::cout << "Error reading: " << ex.what() << std::endl;
            error = true;
        }
    } catch (std::exception& ex) {
        std::cout << "Error initializing: " << ex.what() << std::endl;
        error = true;
    }
    return 0;
}
```

Всё лишнее почистили, чтобы можно было ясно увидеть, как это работает. В начале функции `main()` есть некоторые настройки, используемые для установки соединения – хост, порт, пользователь, пароль. Затем есть вызов инициализации с именем файла бинарного журнала = «mysql-bin». Обратите особое внимание на оператор `setCallback`, который передает имя базы данных = «test», имя таблицы = «test» и адрес функции обратного вызова = callback. Программа войдет в цикл и будет вызывать эту функцию обратного вызова. Посмотрите, как на ранних этапах программы функция обратного вызова выводит «UPDATE», «DELETE» или «INSERT» в зависимости от переданных данных.

ШАГ 5: Поместите программу в директорию `tarantool-libslave` и назовите ее `example.cpp`.

ШАГ 6: Выполните компиляцию и сборку:

```
$ g++ -I/tarantool-libslave/include example.cpp -o example libslave_a.a -ldl -lpthread
```

---

**Примечание:** Замените `tarantool-libslave/include` на полное имя директории.

Обратите внимание, что имя статической библиотеки – `libslave_a.a`, а не `libslave.a`.

---

ШАГ 7: Выполните:

```
$ ./example
```

Результат нет – программа в цикле ожидает, пока сервер MySQL запишет данные в бинарный журнал репликации.

ШАГ 8: Запустите клиентскую программу MySQL – подойдет любая клиентская программа. Введите следующие операторы:

```
USE test
INSERT INTO test VALUES ('A');
INSERT INTO test VALUES ('B');
DELETE FROM test;
```

Проверьте, что происходит в выводе программы `example.cpp` – отображается следующее:

```
INSERT
INSERT
DELETE
DELETE
```

Репликация является построчной, поэтому видим DELETE два раза – потому что есть две строки.

В результате выполнения упражнения видим:

- можно собрать библиотеку, а
- программы, которые используют библиотеку, могут получить доступ ко всему, что сохраняет сервер MySQL.

Более подробную информацию и примеры использования см. ниже:

- Загрузить нашу версию `libslave` можно по ссылке:
  https://github.com/tarantool/libslave
- Ответвление сделано из версии по ссылке (с другим файлом README):
  https://github.com/vozbu/libslave/wiki/API
- Статья How to speed up your MySQL with replication to in-memory database (на английском)
- Статья Репликация из MySQL в Tarantool
- Статья Асинхронная репликация без цензуры

ГЛАВА 4

# Concepts

## 4.1 Data model

Tarantool is a NoSQL database. It stores data in *spaces*, which can be thought of as tables in a relational database, and *tuples*, which are analogous to rows. There are six basic *data operations* in Tarantool.

The platform allows *describing the data schema* but does not require it.

Tarantool supports highly customizable *indexes* of various types.

To ensure *data persistence* and recover quickly in case of failure, Tarantool uses mechanisms like the write-ahead log (WAL) and snapshots.

For details, check the *Data model* page.

## 4.2 Fibers and cooperative multitasking

Tarantool executes code in *fibers* that are managed via *cooperative multitasking*. Learn more about Tarantool's *thread model*.

For details, check the page *Fibers, yields, and cooperative multitasking*.

## 4.3 Transactions

Tarantool's ACID-compliant *transaction model* lets the user choose between two modes of transactions.

The *default mode* allows for fast monopolistic atomic transactions. It doesn't support interactive transactions, and in case of an error, all transaction changes are rolled back.

The *MVCC mode* relies on a multi-version concurrency control engine that allows yielding within a longer transaction. This mode only works with the default in-memory *memtx* storage engine.

For details, check the *Transactions* page.

## 4.4 Application server

Using Tarantool as an application server, you can write applications in Lua, C, or C++. You can also create reusable *modules*.

To increase the speed of code execution, Tarantool has a Lua Just-In-Time compiler (LuaJIT) on board. LuaJIT compiles hot paths in the code – paths that are used many times – thus making the application work faster. To enable developers to work with LuaJIT, Tarantool provides tools like the *memory profiler* and the *getmetrics* module.

For details on Tarantool's modular structure, check the *Modules* page.

To learn how to use Tarantool as an application server, refer to the *guides* in the How-to section.

## 4.5 Sharding

Tarantool implements database sharding via the `vshard` module. For details, go to the *Sharding* page.

## 4.6 Triggers

Tarantool allows specifying callback functions that run upon certain database events. They can be useful for resolving replication conflicts. For details, go to the *Triggers* page.

## 4.7 Replication

Replication allows keeping the data in copies of the same database for better reliability.

Several Tarantool instances can be organized in a replica set. They communicate and transfer data via the *iproto* binary protocol. Learn more about Tarantool's *replication architecture*.

By default, replication in Tarantool is asynchronous. A transaction committed locally on the master node may not get replicated onto other instances before the client receives a success response. Thus, if the master reports success and then dies, the client might not see the result of the transaction.

With *synchronous replication*, transactions on the master node are not considered committed or successful before they are replicated onto a number of instances. This is slower, but more reliable. Synchronous replication in Tarantool is based on an *implementation of the RAFT algorithm*.

For details, check the *Replication* section.

## 4.8 Storage engines

A storage engine is a set of low-level routines that store and retrieve values. Tarantool offers a choice of two storage engines:

- *memtx* is the in-memory storage engine used by default.
- *vinyl* is the on-disk storage engine.

For details, check the *Storage engines* section.

### 4.8.1 Configuration

Tarantool provides the ability to configure the full topology of a cluster and set parameters specific for concrete instances, such as connection settings, memory used to store data, logging, and snapshot settings. Each instance uses this configuration during *startup* to organize the cluster.

There are two approaches to configuring Tarantool:

- *Since version 3.0*: In the YAML format.

  YAML configuration allows you to provide the full cluster topology and specify all configuration options. You can use local configuration in a YAML file for each instance or store configuration data in a reliable *centralized storage*.

- *In version 2.11 and earlier*: *In code* using the `box.cfg` API.

  In this case, configuration is provided in a Lua initialization script.

---

**Примечание:** Starting with the 3.0 version, configuring Tarantool in code is considered a legacy approach.

---

#### Configuration overview

YAML configuration describes the full topology of a Tarantool cluster. A cluster's topology includes the following elements, starting from the lower level:

```
groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            # ...
          instance002:
            # ...
```

- `instances`

  An *instance* represents a single running Tarantool instance. It stores data or might act as a router for handling CRUD requests in a *sharded* cluster.

- `replicasets`

  A *replica set* is a pack of instances that operate on same data sets. *Replication* provides redundancy and increases data availability.

- `groups`

  A *group* provides the ability to organize replica sets. For example, in a sharded cluster, one group can contain *storage* instances and another group can contain *routers* used to handle CRUD requests.

You can flexibly configure a cluster's settings on different levels: from global settings applied to all groups to parameters specific for concrete instances.

### Configuration in a file

This section provides an overview on how to configure Tarantool in a YAML file.

### Basic instance configuration

The example below shows a sample configuration of a single Tarantool instance:

```
groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
```

- The `instances` section includes only one instance named *instance001*. The `iproto.listen.uri` option sets an address used to listen for incoming requests.
- The `replicasets` section contains one replica set named *replicaset001*.
- The `groups` section contains one group named *group001*.

### Configuration scopes

This section shows how to control a scope the specified configuration option is applied to. Most of the configuration options can be applied to a specific instance, replica set, group, or to all instances globally.

- *Instance*

  To apply specific configuration options to a concrete instance, specify such options for this instance only. In the example below, `iproto.listen` is applied to *instance001* only.

  ```
  groups:
    group001:
      replicasets:
        replicaset001:
          instances:
            instance001:
              iproto:
                listen:
                - uri: '127.0.0.1:3301'
  ```

- *Replica set*

  In this example, `iproto.listen` is in effect for all instances in *replicaset001*.

  ```
  groups:
    group001:
      replicasets:
        replicaset001:
          iproto:
  ```

(continues on next page)

(продолжение с предыдущей страницы)

```
          listen:
          - uri: '127.0.0.1:3301'
        instances:
          instance001: { }
```

- *Group*

  In this example, `iproto.listen` is in effect for all instances in *group001*.

```
groups:
  group001:
    iproto:
      listen:
      - uri: '127.0.0.1:3301'
    replicasets:
      replicaset001:
        instances:
          instance001: { }
```

- *Global*

  In this example, `iproto.listen` is applied to all instances of the cluster.

```
iproto:
  listen:
  - uri: '127.0.0.1:3301'

groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001: { }
```

**Примечание:** The *Configuration reference* contains information about scopes to which each configuration option can be applied.

### Configuration scopes: Replica set example

The example below shows how specific configuration options work in different configuration scopes for a replica set with a manual failover. You can learn more about configuring replication from *Replication tutorials*.

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]

iproto:
  advertise:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    peer:
      login: replicator

replication:
  failover: manual

groups:
  group001:
    replicasets:
      replicaset001:
        leader: instance001
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
```

- `credentials` (*global*)

  This section is used to create the *replicator* user and assign it the specified role. These options are applied globally to all instances.

- `iproto` (*global*, *instance*)

  The `iproto` section is specified on both global and instance levels. The `iproto.advertise.peer` option specifies the parameters used by an instance to connect to another instance as a replica, for example, a URI, a login and password, or SSL parameters . In the example above, the option includes `login` only. An URI is taken from `iproto.listen` that is set on the instance level.

- `replication` (*global*)

  The `replication.failover` global option sets a manual failover for all replica sets.

- `leader` (*replica set*)

  The `<replicaset-name>.leader` option sets a *master* instance for *replicaset001*.

### Loading an application

Using Tarantool as an application server, you can run your own Lua applications. In the `app` section, you can load the application and provide a custom application configuration in the `cfg` section.

In the example below, the application is loaded from the `myapp.lua` file placed next to the YAML configuration file:

```
app:
  file: 'myapp.lua'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  cfg:
    greeting: 'Hello'

groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
```

To get a value of the custom `greeting` property in the application code, use the `config:get()` function provided by the *config* module.

```
-- myapp.lua --
local log = require('log').new("myapp")
local config = require('config')
log.info("%s from app, %s!", config:get('app.cfg.greeting'), box.info.name)
```

As a result of *starting* the *instance001*, a log should contain the following line:

```
main/103/interactive/myapp I> Hello from app, instance001!
```

The `app` section can be placed in any *configuration scope*. As an example use case, you can provide different applications for storages and routers in a sharded cluster:

```
groups:
  storages:
    app:
      module: storage
      # ...
  routers:
    app:
      module: router
      # ...
```

Learn more about using Tarantool as the application server from *Developing applications with Tarantool*.

### Predefined variables

In a configuration file, you can use the following predefined variables that are replaced with actual values at runtime:

- `instance_name`
- `replicaset_name`
- `group_name`

To reference these variables in a configuration file, enclose them in double curly braces with whitespaces. In the example below, `{{ instance_name }}` is replaced with *instance001*.

```
groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            snapshot:
              dir: ./var/{{ instance_name }}/snapshots
            wal:
              dir: ./var/{{ instance_name }}/wals
```

As a result, the *paths to snapshots and write-ahead logs* differ for different instances.

### Environment variables

For each configuration parameter, Tarantool provides two sets of predefined environment variables:

- `TT_<CONFIG_PARAMETER>`. These variables are used to substitute parameters specified in a configuration file. This means that these variables have a higher *priority* than the options specified in a configuration file.
- `TT_<CONFIG_PARAMETER>_DEFAULT`. These variables are used to specify default values for parameters missing in a configuration file. These variables have a lower *priority* than the options specified in a configuration file.

For example, `TT_IPROTO_LISTEN` and `TT_IPROTO_LISTEN_DEFAULT` correspond to the `iproto.listen` option. `TT_SNAPSHOT_DIR` and `TT_SNAPSHOT_DIR_DEFAULT` correspond to the `snapshot.dir` option. To see all the supported environment variables, execute the `tarantool` command with the `--help-env-list` *option*.

```
$ tarantool --help-env-list
```

Below are a few examples that show how to set environment variables of different types, like *string*, *number*, *array*, or *map*:

- String. In this example, `TT_LOG_LEVEL` is used to set a logging level to `CRITICAL`:

  ```
  $ export TT_LOG_LEVEL='crit'
  ```

- Number. In this example, a logging level is set to `CRITICAL` using a corresponding numeric value:

  ```
  $ export TT_LOG_LEVEL=3
  ```

- Array. The examples below show how to set the `TT_SHARDING_ROLES` variable that accepts an array value. Arrays can be passed in two ways: using a *simple* ...

  ```
  $ export TT_SHARDING_ROLES=router,storage
  ```

  ... or *JSON* format:

  ```
  $ export TT_SHARDING_ROLES='["router", "storage"]'
  ```

  The *simple* format is applicable only to arrays containing scalar values.

- Map. To assign map values to environment variables, you can also use *simple* or *JSON* formats. In the example below, `TT_LOG_MODULES` sets different logging levels for different modules using a *simple* format:

```
$ export TT_LOG_MODULES=module1=info,module2=error
```

In the next example, `TT_APP_CFG` is used to specify the value of a custom configuration property for a *loaded application* using a *JSON* format:

```
$ export TT_APP_CFG='{"greeting":"Hi"}'
```

The *simple* format is applicable only to maps containing scalar values.

- Array of maps. In the example below, `TT_IPROTO_LISTEN` is used to specify a *listening host and port* values:

```
$ export TT_IPROTO_LISTEN=['{"uri":"127.0.0.1:3311"}']
```

You can also pass several listening addresses:

```
$ export TT_IPROTO_LISTEN=['{"uri":"127.0.0.1:3311"}','{"uri":"127.0.0.1:3312"}']
```

**Примечание:** There are also special `TT_INSTANCE_NAME` and `TT_CONFIG` environment variables that can be used to *start* the specified Tarantool instance with configuration from the given file.

### Centralized configuration

**Enterprise Edition**

Centralized configuration storages are supported by the Enterprise Edition only.

Tarantool enables you to store configuration data in one place using a Tarantool or etcd-based storage. To achieve this, you need to:

1. Set up a centralized configuration storage.
2. Publish a cluster's configuration to the storage.
3. Configure a connection to the storage by providing a local YAML configuration with an endpoint address and key prefix in the `config` section:

```
config:
  etcd:
    endpoints:
    - http://localhost:2379
    prefix: /myapp
```

Learn more from the following guide: *Centralized configuration storages*.

### Configuration precedence

Tarantool configuration options are applied from multiple sources with the following precedence, from highest to lowest:

- `TT_*` *environment variables*.
- Configuration from a *local YAML file*.
- *Centralized configuration*.
- `TT_*_DEFAULT` *environment variables*.

If the same option is defined in two or more locations, the option with the highest precedence is applied.

### Configuration options overview

This section gives an overview of some useful configuration options. All the available options are documented in the *Configuration reference*.

### Connection settings

To configure an address used to listen for incoming requests, use the `iproto.listen` option. The example below shows how to set a listening IP address for `instance001` to `127.0.0.1:3301`:

```
instance001:
  iproto:
    listen:
    - uri: '127.0.0.1:3301'
```

You can learn more from the *Connections* topic.

### Access control

The `credentials` section allows you to create users and grant them the specified privileges. In the example below, a `dbadmin` user with the specified password is created:

```
credentials:
  users:
    dbadmin:
      password: 'T0p_Secret_P@$$w0rd'
```

To learn more, see the *Credentials* section.

### Memory

The *memtx.memory* option specifies how much *memory* Tarantool allocates to actually store data.

```
memtx:
  memory: 1073741824
```

When the limit is reached, `INSERT` or `UPDATE` requests fail with *ER_MEMORY_ISSUE*.

Learn more: *In-memory storage configuration*.

### Snapshots and write-ahead logs

The *snapshot.dir* and *wal.dir* options can be used to configure directories for storing snapshots and write-ahead logs. For example, you can place snapshots and write-ahead logs on different hard drives for better reliability.

```
instance001:
  snapshot:
    dir: '/media/drive1/snapshots'
  wal:
    dir: '/media/drive2/wals'
```

To learn more about the persistence mechanism in Tarantool, see the *Persistence* section. Read more about snapshot and WAL configuration: *Persistence*.

### Centralized configuration storages

---

**Enterprise Edition**

Centralized configuration storages are supported by the Enterprise Edition only.

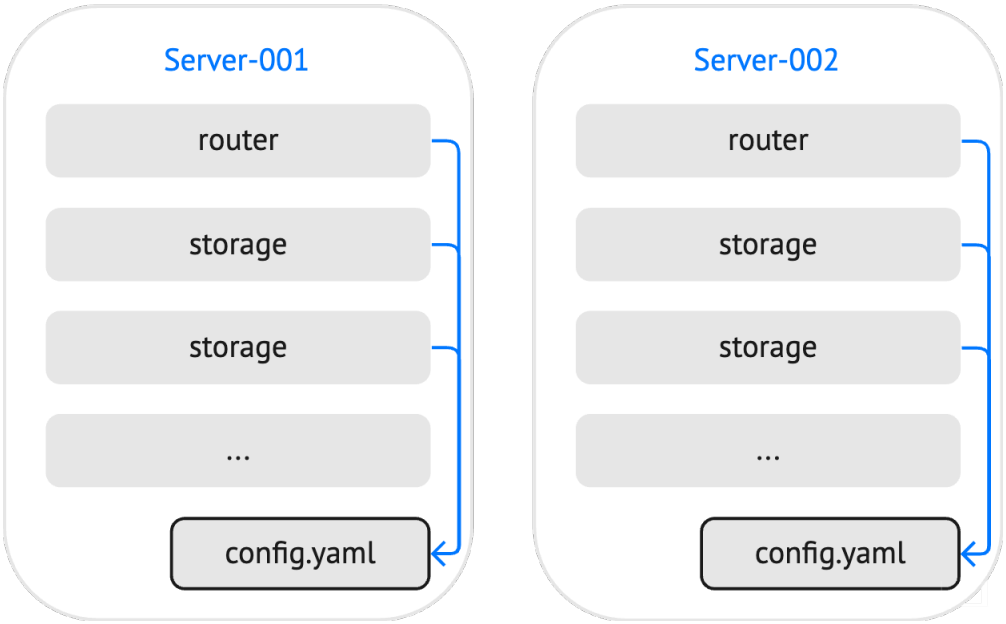
---

**Examples on GitHub**: centralized_config

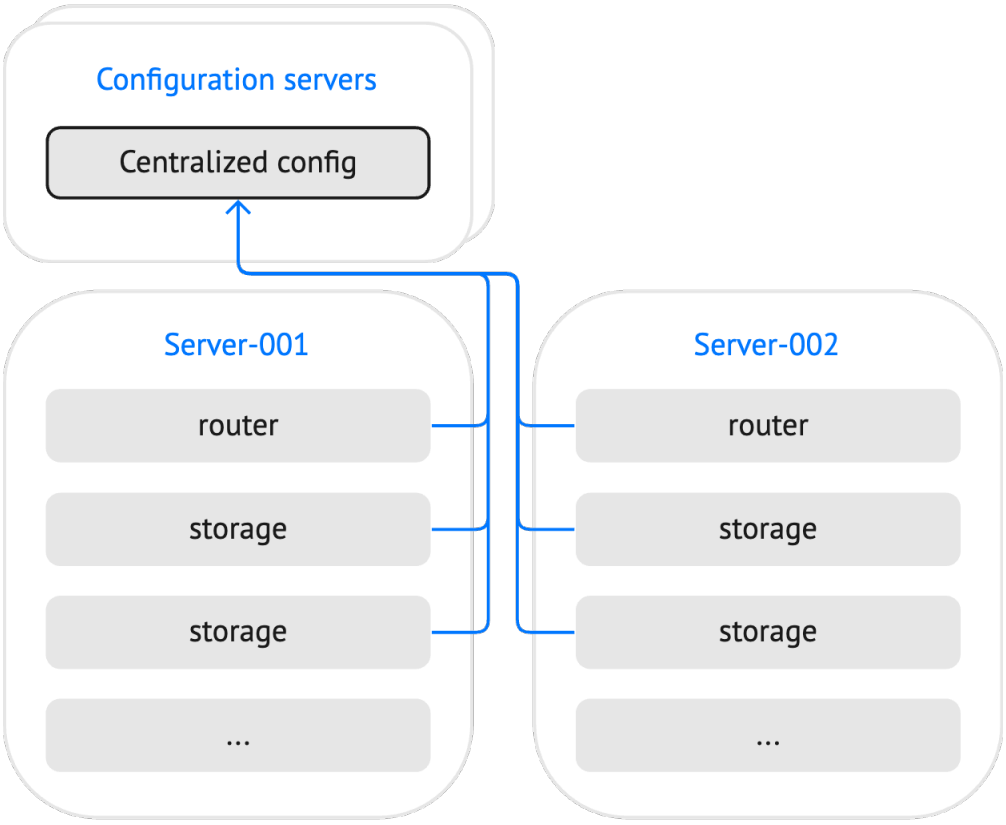
Tarantool enables you to store a cluster's configuration in one reliable place using a Tarantool or etcd-based storage:

- A Tarantool-based configuration storage is a replica set that stores a cluster's configuration in *synchronous* spaces.
- etcd is a distributed key-value storage for any type of critical data used by distributed systems.

With a *local YAML configuration*, you need to make sure that all cluster instances use identical configuration files:




Using a centralized configuration storage, all instances get the actual configuration from one place:




This topic describes how to set up a configuration storage, publish a cluster configuration to this storage, and use this configuration for all cluster instances.

### Setting up a configuration storage

#### Tarantool-based storage

To make a replica set act as a configuration storage, use the built-in `config.storage` *role*.

### Configuring a storage

To *configure* a Tarantool-based storage, follow the steps below:

1. Define a replica set topology and specify the following options at the replica set level:
   - Enable the `config.storage` role in *roles*.
   - Optionally, provide the role configuration in *roles_cfg*. In the example below, the `status_check_interval` option sets the interval (in seconds) of status checks.

```
groups:
  group001:
    replicasets:
      replicaset001:
        roles: [ config.storage ]
        roles_cfg:
          config.storage:
            status_check_interval: 3
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:4401'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:4402'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:4403'
```

2. *Create a user* and grant them the following privileges:

   - The `read` and `write` permissions to the `config_storage` and `config_storage_meta` spaces used to store configuration data.
   - The `execute` permission to `universe` to allow interacting with the storage using the tt utility.

```
credentials:
  users:
    sampleuser:
      password: '123456'
      privileges:
      - permissions: [ read, write ]
        spaces: [ config_storage, config_storage_meta ]
      - permissions: [ execute ]
        universe: true
```

3. Set the *replication.failover* option to `election` to enable automated failover:

```
replication:
  failover: election
```

4. Enable the *MVCC transaction mode* to provide linearizability of read operations:

```
database:
  use_mvcc_engine: true
```

The resulting storage configuration might look as follows:

```
credentials:
  users:
    sampleuser:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      password: '123456'
      privileges:
      - permissions: [ read, write ]
        spaces: [ config_storage, config_storage_meta ]
      - permissions: [ execute ]
        universe: true
    replicator:
      password: 'topsecret'
      roles: [ replication ]

iproto:
  advertise:
    peer:
      login: replicator

replication:
  failover: election

database:
  use_mvcc_engine: true

groups:
  group001:
    replicasets:
      replicaset001:
        roles: [ config.storage ]
        roles_cfg:
          config.storage:
            status_check_interval: 3
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:4401'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:4402'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:4403'
```

You can find the full example here: tarantool_config_storage.

#### Starting a storage

To start instances of the configured storage, use the `tt start` command, for example:

```
$ tt start tarantool_config_storage
```

Learn more from the *Starting and stopping instances* section.

#### etcd-based storage

To learn how to set up an etcd-based configuration storage, consult the etcd documentation.

The example script below demonstrates how to use the `etcdctl` utility to create a user that has read and write access to configurations stored by the `/myapp/` prefix:

```
etcdctl user add root:topsecret
etcdctl role add myapp_config_manager
etcdctl role grant-permission myapp_config_manager --prefix=true readwrite /myapp/
etcdctl user add sampleuser:123456
etcdctl user grant-role sampleuser myapp_config_manager
etcdctl auth enable
```

The credentials of this user should be specified when *configuring a connection* to the etcd cluster.

### Publishing a cluster's configuration

#### Publishing configuration using the tt utility

The tt utility provides the *tt cluster* command for managing centralized cluster configurations. The `tt cluster publish` command can be used to publish a cluster's configuration to both Tarantool and etcd-based storages.

The example below shows how a tt environment and a *layout* of the application called `myapp` might look:

```
├── tt.yaml
├── source.yaml
└── instances.enabled
    └── myapp
        ├── config.yaml
        └── instances.yml
```

- `tt.yaml`: a tt configuration file.
- `source.yaml` contains a cluster's configuration to be published.
- `config.yaml` contains a *local configuration* used to connect to the centralized storage.
- `instances.yml` specifies *instances* to run in the current environment. The configured instances are used by tt when *starting a cluster*. `tt cluster publish` ignores this configuration file.

To publish a cluster's configuration (`source.yaml`) to a centralized storage, execute `tt cluster publish` as follows:

```
$ tt cluster publish "http://sampleuser:123456@localhost:2379/myapp" source.yaml
```

Executing this command publishes a cluster configuration by the `/myapp/config/all` path.

**Примечание:** You can see a cluster's configuration using the `tt cluster show` command.

#### Publishing configuration using the „config“ module

The *config module* provides the API for interacting with a Tarantool-based configuration storage. The example below shows how to read a configuration stored in the `source.yaml` file using the *fio module* API and put this configuration by the `/myapp/config/all` path:

```
local fio = require('fio')
local cluster_config_handle = fio.open('../../source.yaml')
local cluster_config = cluster_config_handle:read()
local response = config.storage.put('/myapp/config/all', cluster_config)
cluster_config_handle:close()
```

Learn more from the *config.storage API* section.

**Примечание:** The `net.box module` provides the ability to monitor configuration updates by watching path or prefix changes. Learn more in *conn:watch()*.

#### Publishing configuration using etcdctl

To publish a cluster's configuration to etcd using the `etcdctl` utility, use the `put` command:

```
$ etcdctl put /myapp/config/all < source.yaml
```

**Примечание:** For etcd versions earlier than 3.4, you need to set the `ETCDCTL_API` environment variable to 3.

### Configuring connection to a storage

To use a configuration from a centralized storage for your cluster, you need to provide connection settings in a local configuration file.

#### Configuring connection to a Tarantool storage

Connection options for a Tarantool-based storage should be specified in the *config.storage* section of the configuration file. In the example below, the following options are specified:

```
config:
  storage:
    endpoints:
      - uri: '127.0.0.1:4401'
        login: sampleuser
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        password: '123456'
      - uri: '127.0.0.1:4402'
        login: sampleuser
        password: '123456'
      - uri: '127.0.0.1:4403'
        login: sampleuser
        password: '123456'
    prefix: /myapp
    timeout: 3
    reconnect_after: 5
```

- `endpoints` specifies the list of configuration storage endpoints.
- `prefix` sets a key prefix used to search a configuration. Tarantool searches keys by the following path: `<prefix>/config/*`. Note that `<prefix>` should start with a slash (/).
- `timeout` specifies the interval (in seconds) to perform the status check of a configuration storage.
- `reconnect_after` specifies how much time to wait (in seconds) before reconnecting to a configuration storage.

You can find the full example here: config_storage.

### Configuring connection to an etcd storage

Connection options for etcd should be specified in the *config.etcd* section of the configuration file. In the example below, the following options are specified:

```
config:
  etcd:
    endpoints:
    - http://localhost:2379
    prefix: /myapp
    username: sampleuser
    password: '123456'
    http:
      request:
        timeout: 3
```

- `endpoints` specifies the list of etcd endpoints.
- `prefix` sets a key prefix used to search a configuration. Tarantool searches keys by the following path: `<prefix>/config/*`. Note that `<prefix>` should start with a slash (/).
- `username` and `password` specify credentials used for authentication.
- `http.request.timeout` configures a request timeout for an etcd server.

You can find the full example here: config_etcd.

### Starting a cluster

The *tt* utility is the recommended way to start Tarantool instances. You can learn how to do this from the *Starting and stopping instances* section.

You can also use the `tarantool` command to *start a Tarantool instance*. In this case, you can eliminate creating a *local configuration* and provide connection settings using the following *environment variables*:

- Tarantool-based storage: `TT_CONFIG_STORAGE_ENDPOINTS` and `TT_CONFIG_STORAGE_PREFIX`.
- etcd-based storage: `TT_CONFIG_ETCD_ENDPOINTS` and `TT_CONFIG_ETCD_PREFIX`.

The example below shows how to provide etcd connection settings and start cluster instances using the `tarantool` command:

```
$ export TT_CONFIG_ETCD_ENDPOINTS=http://localhost:2379
$ export TT_CONFIG_ETCD_PREFIX=/myapp

$ tarantool --name instance001
$ tarantool --name instance002
$ tarantool --name instance003
```

### Reloading configuration

By default, Tarantool watches keys with the *specified prefix* for changes in a cluster's configuration and reloads a changed configuration automatically. If necessary, you can set the *config.reload* option to `manual` to turn off configuration reloading:

```
config:
  reload: 'manual'
  etcd:
    # ...
```

In this case, you can reload a configuration in an *admin console* or *application code* using the *reload()* function provided by the `config` module:

```
require('config'):reload()
```

### Configuration in code

---

**Примечание:** Starting with the 3.0 version, the recommended way of configuring Tarantool is using a *configuration file*. Configuring Tarantool in code is considered a legacy approach.

---

This topic covers the specifics of configuring Tarantool in code using the `box.cfg` API. In this case, a configuration is stored in an *initialization file* - a Lua script with the specified configuration options. You can find all the available options in the *Configuration reference*.

### Initialization file

If the command to *start Tarantool* includes an instance file, then Tarantool begins by invoking the Lua program in the file, which may have the name `init.lua`. The Lua program may get further arguments from the command line or may use operating-system functions, such as `getenv()`. The Lua program almost always begins by invoking `box.cfg()`, if the database server will be used or if ports need to be opened. For example, suppose `init.lua` contains the lines

```
#!/usr/bin/env tarantool
box.cfg{
    listen              = os.getenv("LISTEN_URI"),
    memtx_memory        = 33554432,
    pid_file            = "tarantool.pid",
    wal_max_size        = 2500
}
print('Starting ', arg[1])
```

and suppose the environment variable `LISTEN_URI` contains 3301, and suppose the command line is `tarantool init.lua ARG`. Then the screen might look like this:

```
$ export LISTEN_URI=3301
$ tarantool init.lua ARG
... main/101/init.lua C> Tarantool 2.8.3-0-g01023dbc2
... main/101/init.lua C> log level 5
... main/101/init.lua I> mapping 33554432 bytes for memtx tuple arena...
... main/101/init.lua I> recovery start
... main/101/init.lua I> recovering from './00000000000000000000.snap'
... main/101/init.lua I> set 'listen' configuration option to "3301"
... main/102/leave_local_hot_standby I> ready to accept requests
Starting  ARG
... main C> entering the event loop
```

If you wish to start an interactive session on the same terminal after initialization is complete, you can pass the `-i` *command-line option*.

### Environment variables

Starting from version *2.8.1*, you can specify configuration parameters via special environment variables. The name of a variable should have the following pattern: `TT_<NAME>`, where `<NAME>` is the uppercase name of the corresponding *box.cfg parameter*.

For example:

- `TT_LISTEN` – corresponds to the *box.cfg.listen* option.
- `TT_MEMTX_DIR` – corresponds to the *box.cfg.memtx_dir* option.

In case of an array value, separate the array elements by a comma without space:

```
export TT_REPLICATION="localhost:3301,localhost:3302"
```

If you need to pass *additional parameters for URI*, use the `?` and `&` delimiters:

```
export TT_LISTEN="localhost:3301?param1=value1&param2=value2"
```

An empty variable (`TT_LISTEN=`) has the same effect as an unset one, meaning that the corresponding configuration parameter won't be set when calling `box.cfg{}`.

### Configuration parameters

Configuration parameters have the form:

`box.cfg{[key = value [, key = value ...]]}`

Configuration parameters can be set in a Lua *initialization file*, which is specified on the Tarantool command line.

Most configuration parameters are for allocating resources, opening ports, and specifying database behavior. All parameters are optional. Most of the parameters are dynamic, that is, they can be changed at runtime by calling `box.cfg{}` a second time. For example, the command below sets the *listen port* to `3301`.

```
tarantool> box.cfg{ listen = 3301 }
2023-05-10 13:28:54.667 [31326] main/103/interactive I> tx_binary: stopped
2023-05-10 13:28:54.667 [31326] main/103/interactive I> tx_binary: bound to [::]:3301
2023-05-10 13:28:54.667 [31326] main/103/interactive/box.load_cfg I> set 'listen'␣
↪configuration option to 3301
---
...
```

To see all the non-null parameters, execute `box.cfg` (no parentheses).

```
tarantool> box.cfg
---
- replication_skip_conflict: false
  wal_queue_max_size: 16777216
  feedback_host: https://feedback.tarantool.io
  memtx_dir: .
  memtx_min_tuple_size: 16
  -- other parameters --
...
```

To see a particular parameter value, call a corresponding `box.cfg` option. For example, `box.cfg.listen` shows the specified *listen address*.

```
tarantool> box.cfg.listen
---
- 3301
...
```

### Listen URI

Some configuration parameters and some functions depend on a URI (Universal Resource Identifier). The URI string format is similar to the generic syntax for a URI schema. It may contain (in order):

- user name for login
- password
- host name or host IP address
- port number

- query parameters

Only a port number is always mandatory. A password is mandatory if a user name is specified unless the user name is „guest“.

Formally, the URI syntax is `[host:]port` or `[username:password@]host:port`. If a host is omitted, then «0.0.0.0» or «[::]» is assumed, meaning respectively any IPv4 address or any IPv6 address on the local machine. If `username:password` is omitted, then the «guest» user is assumed. Some examples:

| URI fragment | Example |
|---|---|
| port | 3301 |
| host:port | 127.0.0.1:3301 |
| username:password@host:port | notguest:sesame@mail.ru:3301 |

In code, the URI value can be passed as a number (if only a port is specified) or a string:

```
box.cfg { listen = 3301 }

box.cfg { listen = "127.0.0.1:3301" }
```

In certain circumstances, a Unix domain socket may be used where a URI is expected, for example, `unix/:/tmp/unix_domain_socket.sock` or simply `/tmp/unix_domain_socket.sock`.

The *uri* module provides functions that convert URI strings into their components or turn components into URI strings.

### Specifying several URIs

Starting from version 2.10.0, a user can open several listening iproto sockets on a Tarantool instance and, consequently, can specify several URIs in the configuration parameters such as *box.cfg.listen* and *box.cfg.replication*.

URI values can be set in a number of ways:

- As a string with URI values separated by commas.

  ```
  box.cfg { listen = "127.0.0.1:3301, /unix.sock, 3302" }
  ```

- As a table that contains URIs in the string format.

  ```
  box.cfg { listen = {"127.0.0.1:3301", "/unix.sock", "3302"} }
  ```

- As an array of tables with the `uri` field.

  ```
  box.cfg { listen = {
          {uri = "127.0.0.1:3301"},
          {uri = "/unix.sock"},
          {uri = 3302}
      }
  }
  ```

- In a combined way – an array that contains URIs in both the string and the table formats.

```
box.cfg { listen = {
        "127.0.0.1:3301",
        { uri = "/unix.sock" },
        { uri = 3302 }
    }
}
```

Also, starting from version 2.10.0, it is possible to specify additional parameters for URIs. You can do this in different ways:

- Using the `?` delimiter when URIs are specified in a string format.

```
box.cfg { listen = "127.0.0.1:3301?p1=value1&p2=value2, /unix.sock?p3=value3" }
```

- Using the `params` table: a URI is passed in a table with additional parameters in the «params» table. Parameters in the «params» table overwrite the ones from a URI string («value2» overwrites «value1» for `p1` in the example below).

```
box.cfg { listen = {
        "127.0.0.1:3301?p1=value1",
        params = {p1 = "value2", p2 = "value3"}
    }
}
```

- Using the `default_params` table for specifying default parameter values.

  In the example below, two URIs are passed in a table. The default value for the `p3` parameter is defined in the `default_params` table and used if this parameter is not specified in URIs. Parameters in the `default_params` table are applicable to all the URIs passed in a table.

```
box.cfg { listen = {
        "127.0.0.1:3301?p1=value1",
        { uri = "/unix.sock", params = { p2 = "value2" } },
        default_params = { p3 = "value3" }
    }
}
```

The recommended way for specifying URI with additional parameters is the following:

```
box.cfg { listen = {
        {uri = "127.0.0.1:3301", params = {p1 = "value1"}},
        {uri = "/unix.sock", params = {p2 = "value2"}},
        {uri = 3302, params = {p3 = "value3"}}
    }
}
```

In case of a single URI, the following syntax also works:

```
box.cfg { listen = {
        uri = "127.0.0.1:3301",
        params = { p1 = "value1", p2 = "value2" }
    }
}
```

### Traffic encryption

**Enterprise Edition**

Traffic encryption is supported by the Enterprise Edition only.

Since version 2.10.0, Tarantool Enterprise Edition has the built-in support for using SSL to encrypt the client-server communications over *binary connections*, that is, between Tarantool instances in a cluster or connecting to an instance via connectors using *net.box*.

Tarantool uses the OpenSSL library that is included in the delivery package. Note that SSL connections use only TLSv1.2.

#### Configuration

To configure traffic encryption, you need to set the special *URI parameters* for a particular connection. The parameters can be set for the following `box.cfg` options and `net.box` method:

- *box.cfg.listen* – on the server side.
- *box.cfg.replication* – on the client side.
- *net_box_object.connect()* – on the client side.

Below is the list of the parameters. In the *next section*, you can find details and examples on what should be configured on both the server side and the client side.

- `transport` – enables SSL encryption for a connection if set to `ssl`. The default value is `plain`, which means the encryption is off. If the parameter is not set, the encryption is off too. Other encryption-related parameters can be used only if the `transport = 'ssl'` is set.

  Example:

  ```
  local connection = require('net.box').connect({
      uri = 'admin:topsecret@127.0.0.1:3301',
      params = { transport = 'ssl',
                 ssl_cert_file = 'certs/instance001/server001.crt',
                 ssl_key_file = 'certs/instance001/server001.key',
                 ssl_password = 'qwerty' }
  })
  ```

- `ssl_key_file` – a path to a private SSL key file. Mandatory for a server. For a client, it's mandatory if the `ssl_ca_file` parameter is set for a server; otherwise, optional. If the private key is encrypted, provide a password for it in the `ssl_password` or `ssl_password_file` parameter.
- `ssl_cert_file` – a path to an SSL certificate file. Mandatory for a server. For a client, it's mandatory if the `ssl_ca_file` parameter is set for a server; otherwise, optional.
- `ssl_ca_file` – a path to a trusted certificate authorities (CA) file. Optional. If not set, the peer won't be checked for authenticity.

  Both a server and a client can use the `ssl_ca_file` parameter:

  – If it's on the server side, the server verifies the client.

  – If it's on the client side, the client verifies the server.

  – If both sides have the CA files, the server and the client verify each other.

- `ssl_ciphers` – a colon-separated (`:`) list of SSL cipher suites the connection can use. See the *Supported ciphers* section for details. Optional. Note that the list is not validated: if a cipher suite is unknown, Tarantool just ignores it, doesn't establish the connection and writes to the log that no shared cipher found.
- `ssl_password` – a password for an encrypted private SSL key. Optional. Alternatively, the password can be provided in `ssl_password_file`.
- `ssl_password_file` – a text file with one or more passwords for encrypted private SSL keys (each on a separate line). Optional. Alternatively, the password can be provided in `ssl_password`.

  Tarantool applies the `ssl_password` and `ssl_password_file` parameters in the following order:

  1. If `ssl_password` is provided, Tarantool tries to decrypt the private key with it.
  2. If `ssl_password` is incorrect or isn't provided, Tarantool tries all passwords from `ssl_password_file` one by one in the order they are written.
  3. If `ssl_password` and all passwords from `ssl_password_file` are incorrect, or none of them is provided, Tarantool treats the private key as unencrypted.

Configuration example:

```
box.cfg{ listen = {
    uri = 'localhost:3301',
    params = {
        transport = 'ssl',
        ssl_key_file = '/path_to_key_file',
        ssl_cert_file = '/path_to_cert_file',
        ssl_ciphers = 'HIGH:!aNULL',
        ssl_password = 'topsecret'
    }
}}
```

### Supported ciphers

Tarantool Enterprise supports the following cipher suites:

- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-RSA-AES256-GCM-SHA384
- DHE-RSA-AES256-GCM-SHA384
- ECDHE-ECDSA-CHACHA20-POLY1305
- ECDHE-RSA-CHACHA20-POLY1305
- DHE-RSA-CHACHA20-POLY1305
- ECDHE-ECDSA-AES128-GCM-SHA256
- ECDHE-RSA-AES128-GCM-SHA256
- DHE-RSA-AES128-GCM-SHA256
- ECDHE-ECDSA-AES256-SHA384
- ECDHE-RSA-AES256-SHA384
- DHE-RSA-AES256-SHA256
- ECDHE-ECDSA-AES128-SHA256

- ECDHE-RSA-AES128-SHA256
- DHE-RSA-AES128-SHA256
- ECDHE-ECDSA-AES256-SHA
- ECDHE-RSA-AES256-SHA
- DHE-RSA-AES256-SHA
- ECDHE-ECDSA-AES128-SHA
- ECDHE-RSA-AES128-SHA
- DHE-RSA-AES128-SHA
- AES256-GCM-SHA384
- AES128-GCM-SHA256
- AES256-SHA256
- AES128-SHA256
- AES256-SHA
- AES128-SHA
- GOST2012-GOST8912-GOST8912
- GOST2001-GOST89-GOST89

Tarantool Enterprise static build has the embedded engine to support the GOST cryptographic algorithms. If you use these algorithms for traffic encryption, specify the corresponding cipher suite in the `ssl_ciphers` parameter, for example:

```
box.cfg{ listen = {
    uri = 'localhost:3301',
    params = {
        transport = 'ssl',
        ssl_key_file = '/path_to_key_file',
        ssl_cert_file = '/path_to_cert_file',
        ssl_ciphers = 'GOST2012-GOST8912-GOST8912'
    }
}}
```

For detailed information on SSL ciphers and their syntax, refer to OpenSSL documentation.

### Using environment variables

The URI parameters for traffic encryption can also be set via *environment variables*, for example:

```
export TT_LISTEN="localhost:3301?transport=ssl&ssl_cert_file=/path_to_cert_file&ssl_key_
↪file=/path_to_key_file"
```

### Server-client configuration details

When configuring the traffic encryption, you need to specify the necessary parameters on both the server side and the client side. Below you can find the summary on the options and parameters to be used and *examples of configuration*.

**Server side**

- Is configured via the `box.cfg.listen` option.
- Mandatory URI parameters: `transport`, `ssl_key_file` and `ssl_cert_file`.
- Optional URI parameters: `ssl_ca_file`, `ssl_ciphers`, `ssl_password`, and `ssl_password_file`.

**Client side**

- Is configured via the `box.cfg.replication` option (see *details*) or `net_box_object.connect()`.

Parameters:

- If the server side has only the `transport`, `ssl_key_file` and `ssl_cert_file` parameters set, on the client side, you need to specify only `transport = ssl` as the mandatory parameter. All other URI parameters are optional.
- If the server side also has the `ssl_ca_file` parameter set, on the client side, you need to specify `transport`, `ssl_key_file` and `ssl_cert_file` as the mandatory parameters. Other parameters – `ssl_ca_file`, `ssl_ciphers`, `ssl_password`, and `ssl_password_file` – are optional.

### Configuration examples

Suppose, there is a *master-replica* set with two Tarantool instances:

- 127.0.0.1:3301 – master (server)
- 127.0.0.1:3302 – replica (client).

Examples below show the configuration related to connection encryption for two cases: when the trusted certificate authorities (CA) file is not set on the server side and when it does. Only mandatory URI parameters are mentioned in these examples.

1. **Without CA**

- 127.0.0.1:3301 – master (server)

```
box.cfg{
    listen = {
        uri = '127.0.0.1:3301',
        params = {
            transport = 'ssl',
            ssl_key_file = '/path_to_key_file',
            ssl_cert_file = '/path_to_cert_file'
        }
    }
}
```

- 127.0.0.1:3302 – replica (client)

```
box.cfg{
    listen = {
        uri = '127.0.0.1:3302',
        params = {transport = 'ssl'}
    },
    replication = {
        uri = 'username:password@127.0.0.1:3301',
        params = {transport = 'ssl'}
    },
    read_only = true
}
```

2. **With CA**

- 127.0.0.1:3301 – master (server)

```
box.cfg{
    listen = {
        uri = '127.0.0.1:3301',
        params = {
            transport = 'ssl',
            ssl_key_file = '/path_to_key_file',
            ssl_cert_file = '/path_to_cert_file',
            ssl_ca_file = '/path_to_ca_file'
        }
    }
}
```

- 127.0.0.1:3302 – replica (client)

```
box.cfg{
    listen = {
        uri = '127.0.0.1:3302',
        params = {
            transport = 'ssl',
            ssl_key_file = '/path_to_key_file',
            ssl_cert_file = '/path_to_cert_file'
        }
    },
    replication = {
        uri = 'username:password@127.0.0.1:3301',
        params = {
            transport = 'ssl',
            ssl_key_file = '/path_to_key_file',
            ssl_cert_file = '/path_to_cert_file'
        }
    },
    read_only = true
}
```

### Starting a Tarantool instance

Below is the syntax for starting a Tarantool instance configured in a Lua initialization script:

```
$ tarantool LUA_INITIALIZATION_FILE [OPTION ...]
```

The `tarantool` command also provides a set of *options* that might be helpful for development purposes.

The command below starts a Tarantool instance configured in the `init.lua` file:

```
$ tarantool init.lua
```

### In-memory storage

**Example on GitHub**: memtx

In Tarantool, all data is stored in random-access memory (RAM) by default. For this purpose, the *memtx* storage engine is used.

This topic describes how to define basic settings related to in-memory storage in the *memtx* section of a *YAML configuration* – for example, *memory size* and *maximum tuple size*. For the specific settings related to allocator or sorting threads, check the corresponding `memtx` options in the *Configuration reference*.

---

**Примечание:** To estimate the required amount of memory, you can use the sizing calculator.

---

### Memory size

In Tarantool, data is stored in spaces. Each space consists of tuples – the database records. To specify the amount of memory that Tarantool allocates to store tuples, use the *memtx.memory* configuration option.

In the example below, the memory size is set to 1 GB (1073741824 bytes):

```
memtx:
  memory: 1073741824
```

The server does not exceed this limit to allocate tuples. For indexes and connection information, additional memory is used.

When the `memtx.memory` limit is reached, `INSERT` or `UPDATE` requests fail with *ER_MEMORY_ISSUE*.

### Tuple size

You can configure the minimum and the maximum tuple sizes in bytes.

- If the tuples are small, you can decrease the minimum size.
- If the tuples are large, you can increase the maximum size.

To define the tuple size, use the *memtx.min_tuple_size* and *memtx.max_tuple_size* configuration options.

In the example, the minimum size is set to 8 bytes and the maximum size is set to 5 MB:

```
memtx:
  memory: 1073741824
  min_tuple_size: 8
  max_tuple_size: 5242880
```

### Persistence

To ensure data persistence, Tarantool provides the abilities to:

- Record each data change request into a *write-ahead log* (WAL) file (`.xlog` files).
- Take *snapshots* that contain an on-disk copy of the entire data set for a given moment (`.snap` files). It is possible to set automatic snapshot creation using the *checkpoint daemon*.

During the recovery process, Tarantool can load the latest snapshot file and then read the requests from the WAL files, produced after this snapshot was made. This topic describes how to configure:

- the snapshot creation in the *snapshot* section of a *YAML configuration*.
- the recording to the write-ahead log in the *wal* section of a YAML configuration.

To learn more about the persistence mechanism in Tarantool, see the *Persistence* section. The formats of WAL and snapshot files are described in detail in the *File formats* section.

### Configure the snapshots

**Example on GitHub**: snapshot

This section describes how to define snapshot settings in the *snapshot* section of a YAML configuration.

### Set up automatic snapshot creation

In Tarantool, it is possible to automate the snapshot creation. Automatic creation is enabled by default and can be configured in two ways:

- A new snapshot is taken once in a given period (see *snapshot.by.interval*).
- A new snapshot is taken once the size of all WAL files created since the last snapshot exceeds a given limit (see *snapshot.by.wal_ size*).

The `snapshot.by.interval` option sets up the *checkpoint daemon* that takes a new snapshot every `snapshot.by.interval` seconds. If the `snapshot.by.interval` option is set to zero, the checkpoint daemon is disabled.

The `snapshot.by.wal_size` option defines the maximum size in bytes for all WAL files created since the last snapshot taken. Once this size is exceeded, the checkpoint daemon takes a snapshot. Then, *Tarantool garbage collector* deletes the old WAL files.

The example shows how to specify the `snapshot.by.interval` and the `snapshot.by.wal_size` options:

```
by:
  interval: 7200
  wal_size: 1000000000000000000
```

In the example, a new snapshot is created in two cases:

- every 2 hours (every 7200 seconds)

- when the size for all WAL files created since the last snapshot reaches the size of 1e18 (1000000000000000000) bytes.

### Specify a directory for snapshot files

To configure a directory where the snapshot files are stored, use the *snapshot.dir* configuration option. The example below shows how to specify a snapshot directory for `instance001` explicitly:

```
instance001:
  snapshot:
    dir: 'var/lib/{{ instance_name }}/snapshots'
```

By default, WAL files and snapshot files are stored in the same directory `var/lib/{{ instance_name }}`. However, you can specify different directories for them. For example, you can place snapshots and write-ahead logs on different hard drives for better reliability:

```
instance001:
  snapshot:
    dir: '/media/drive1/snapshots'
  wal:
    dir: '/media/drive2/wals'
```

### Configure a maximum number of stored snapshots

You can set a limit on the number of snapshots stored in the *snapshot.dir* directory using the *snapshot.count* option. Once the number of snapshots reaches the given limit, *Tarantool garbage collector* deletes the oldest snapshot file and any associated WAL files after the new snapshot is taken.

In the example below, the snapshot is created every two hours (every 7200 seconds) until there are three snapshots in the `snapshot.dir` directory. After creating a new snapshot (the fourth one), the oldest snapshot and the corresponding WALs are deleted.

```
count: 3
by:
  interval: 7200
```

### Configure the write-ahead log

**Example on GitHub**: wal

This section describes how to define WAL settings in the *wal* section of a YAML configuration.

### Set the WAL mode

The recording to the write-ahead log is enabled by default. It means that if an instance restart occurs, the data will be recovered. The recording to the WAL can be configured using the *wal.mode* configuration option.

There are two modes that enable writing to the WAL:

- `write` (default) – enable WAL and write the data without waiting for the data to be flushed to the storage device.
- `fsync` – enable WAL and ensure that the record is written to the storage device.

The example below shows how to specify the `write` WAL mode:

```
mode: 'write'
```

To turn the WAL writer off, set the `wal.mode` option to `none`.

### Specify a directory for WAL files

To configure a directory where the WAL files are stored, use the *wal.dir* configuration option. The example below shows how to specify a directory for `instance001` explicitly:

```
instance001:
  wal:
    dir: 'var/lib/{{ instance_name }}/wals'
```

### Set an interval between scans

In case of *replication* or *hot standby* mode, Tarantool scans for changes in the WAL files every *wal.dir_rescan_delay* seconds. The example below shows how to specify the interval between scans:

```
dir_rescan_delay: 3
```

### Set a maximum size for the WAL file

A new WAL file is created when the current one reaches the *wal.max_size* size. The configuration for this option might look as follows:

```
max_size: 268435456
```

### Set a delay for the garbage collector

In Tarantool, the *checkpoint daemon* takes new snapshots at the given interval (see *snapshot.by.interval*). After an instance restart, the Tarantool garbage collector deletes the old WAL files.

To delay the immediate deletion of WAL files, use the *wal.cleanup_delay* configuration option. The delay eliminates possible erroneous situations when the master deletes WALs needed by *replicas* after restart. As a consequence, replicas sync with the master faster after its restart and don't need to download all the data again.

In the example, the delay is set to 5 hours (18000 seconds):

```
cleanup_delay: 18000
```

### Specify the WAL extensions

In Tarantool Enterprise, you can store an old and new tuple for each CRUD operation performed. A detailed description and examples of the WAL extensions are provided in the *WAL extensions* section.

See also: *wal.ext.** configuration options.

### Checkpoint daemon

The checkpoint daemon (snapshot daemon) is a constantly running *fiber*. The checkpoint daemon creates a schedule for the periodic snapshot creation based on the *configuration options* and the speed of file size growth. If enabled, the daemon makes new *snapshot* (`.snap`) files according to this schedule.

The work of the checkpoint daemon is based on the following configuration options:

- *snapshot.by.interval* – a new snapshot is taken once in a given period.
- *snapshot.by.wal_size* – a new snapshot is taken once the size of all WAL files created since the last snapshot exceeds a given limit.

If necessary, the checkpoint daemon also activates the *Tarantool garbage collector* that deletes old snapshots and WAL files.

### Tarantool garbage collector

Tarantool garbage collector can be activated by the *checkpoint daemon*. The garbage collector tracks the snapshots that are to be *relayed to a replica* or needed by other consumers. When the files are no longer needed, Tarantool garbage collector deletes them.

---

**Примечание:** The garbage collector called by the checkpoint daemon is distinct from the Lua garbage collector which is for Lua objects, and distinct from the Tarantool garbage collector that specializes in *handling shard buckets*.

---

This garbage collector is called as follows:

- When the number of snapshots reaches the limit of *snapshot.count* size. After a new snapshot is taken, Tarantool garbage collector deletes the oldest snapshot file and any associated WAL files.
- When the size of all WAL files created since the last snapshot reaches the limit of *snapshot.by.wal_size*. Once this size is exceeded, the checkpoint daemon takes a snapshot, then the garbage collector deletes the old WAL files.

If an old snapshot file is deleted, the Tarantool garbage collector also deletes any *write-ahead log (.xlog)* files that meet the following conditions:

- The WAL files are older than the snapshot file.
- The WAL files contain information present in the snapshot file.

Tarantool garbage collector also deletes obsolete vinyl `.run` files.

Tarantool garbage collector doesn't delete a file in the following cases:

- A backup is running, and the file has not been backed up (see *Hot backup*).
- Replication is running, and the file has not been relayed to a replica (see *Replication architecture*),
- A replica is connecting.
- A replica has fallen behind. The progress of each replica is tracked; if a replica's position is far from being up to date, then the server stops to give it a chance to catch up. If an administrator concludes that a replica is permanently down, then the correct procedure is to restart the server, or (preferably) *remove the replica from the cluster*.

### Connections

To set up a Tarantool cluster, you need to enable communication between its instances, regardless of whether they running on one or different hosts. This requires *configuring* connection settings that include:

- One or several URIs used to listen for incoming requests.
- An URI used to advertise an instance to other cluster members. This URI lets other cluster members know how to connect to the current Tarantool instance.
- (Optional) SSL settings used to secure connections between instances.

Configuring connection settings is also required to enable communication of a Tarantool cluster to external systems. For example, this might be administering cluster members using *tt*, managing clusters using *Tarantool Cluster Manager*, or using *connectors* for different languages.

This topic describes how to define connection settings in the *iproto* section of a YAML configuration.

---

**Примечание:** iproto is a *binary protocol* used to communicate between cluster instances and with external systems.

---

### Listen URI

To configure URIs used to listen for incoming requests, use the *iproto.listen* configuration option.

### One listen address

The example below shows how to set a listening IP address for `instance001` to `127.0.0.1:3301`:

```
instance001:
  iproto:
    listen:
    - uri: '127.0.0.1:3301'
```

#### Multiple listen addresses

In this example, `instance001` listens on two IP addresses:

```
instance001:
  iproto:
    listen:
    - uri: '127.0.0.1:3301'
    - uri: '127.0.0.1:3302'
```

#### Listen port

You can pass only a port value to `iproto.listen`:

```
instance001:
  iproto:
    listen:
    - uri: '3301'
```

In this case, this port is used for all IP addresses the server listens on.

#### SSL parameters

In the Enterprise Edition, you can enable SSL for a connection using the `params` section of the specified URI:

```
instance001:
  iproto:
    listen:
    - uri: '127.0.0.1:3301'
      params:
        transport: 'ssl'
        ssl_cert_file: 'certs/server.crt'
        ssl_key_file: 'certs/server.key'
```

Learn more from *Securing connections with SSL*.

#### Unix domain socket

For local development, you can enable communication between cluster members by using Unix domain sockets:

```
instance001:
  iproto:
    listen:
    - uri: 'unix/:../var/run/{{ instance_name }}/tarantool.iproto'
```

### Advertise URI

An advertise URI (*iproto.advertise.**) lets other cluster members or clients know how to connect to the current Tarantool instance:

- `iproto.advertise.peer` specifies how to advertise the instance to other cluster members.
- `iproto.advertise.sharding` specifies how to advertise the instance to a *router* and *rebalancer*.
- `iproto.advertise.client` accepts a URI used to advertise the instance to clients.

`iproto.advertise.<peer_or_sharding>` might include the credentials required to connect to this instance, a URI used to listen for incoming requests, and SSL settings.

If `iproto.advertise.<peer_or_sharding>.uri` is not specified explicitly, a *listen URI* of this instance is used. In this case, you need at least to specify credentials for connecting to this instance.

### Connection credentials

In the example below, the `iproto.advertise.peer` option is used to inform other replica set members that the `replicator` user should be used to connect to the current instance:

```
iproto:
  advertise:
    peer:
      login: replicator
```

In a sharded cluster, `iproto.advertise.sharding` specifies that a router and rebalancer should use the `storage` user to connect to storages:

```
iproto:
  advertise:
    peer:
      login: replicator
    sharding:
      login: storage
```

### URI

If required, you can specify an advertise URI explicitly by setting up the *iproto.advertise.<peer_or_sharding>.uri* option. In the example below, `iproto.listen` includes two URIs that can be used to connect to `instance001` but only the second one is used to advertise this instance to other replica set peers:

```
instance001:
  iproto:
    listen:
    - uri: '127.0.0.1:3301'
    - uri: '127.0.0.1:4401'
    advertise:
      peer:
        uri: '127.0.0.1:4401'
```

The `iproto.advertise.<peer_or_sharding>.uri` option can also accept an FQDN instead of an IP address:

```
instance001:
  iproto:
    listen:
    - uri: '192.168.0.101:3301'
    advertise:
      peer:
        uri: 'server001.example.com:3301'
```

To learn about the specifics of configuring an advertise URI's SSL settings, see *Advertise URI specifics*.

### Securing connections with SSL

---

**Enterprise Edition**

SSL is supported by the Enterprise Edition only.

---

Tarantool supports the use of SSL connections to encrypt client-server communications for increased security. To enable SSL, use the *<uri>.params.** options, which can be applied to both listen and advertise URIs.

#### Without CA

The example below demonstrates how to enable traffic encryption by using a self-signed server certificate. The following parameters are specified for each instance:

- *ssl_cert_file*: a path to an SSL certificate file.
- *ssl_key_file*: a path to a private SSL key file.

```
instances:
  instance001:
    iproto:
      listen:
      - uri: '127.0.0.1:3301'
        params:
          transport: 'ssl'
          ssl_cert_file: 'certs/server.crt'
          ssl_key_file: 'certs/server.key'
  instance002:
    iproto:
      listen:
      - uri: '127.0.0.1:3302'
        params:
          transport: 'ssl'
          ssl_cert_file: 'certs/server.crt'
          ssl_key_file: 'certs/server.key'
  instance003:
    iproto:
      listen:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      - uri: '127.0.0.1:3303'
        params:
          transport: 'ssl'
          ssl_cert_file: 'certs/server.crt'
          ssl_key_file: 'certs/server.key'
```

You can find the full example here: ssl_without_ca.

### With CA

The example below demonstrates how to enable traffic encryption by using a server certificate signed by a trusted certificate authority. In this case, all replica set peers verify each other for authenticity.

The following parameters are specified for each instance:

- *ssl_ca_file*: a path to a trusted certificate authorities (CA) file.
- *ssl_cert_file*: a path to an SSL certificate file.
- *ssl_key_file*: a path to a private SSL key file.
- *ssl_password* (`instance001`): a password for an encrypted private SSL key.
- *ssl_password_file* (`instance002` and `instance003`): a text file containing passwords for encrypted SSL keys.
- *ssl_ciphers*: a colon-separated list of SSL cipher suites the connection can use.

```
instances:
  instance001:
    iproto:
      listen:
      - uri: '127.0.0.1:3301'
        params:
          transport: 'ssl'
          ssl_ca_file: 'certs/root_ca.crt'
          ssl_cert_file: 'certs/instance001/server001.crt'
          ssl_key_file: 'certs/instance001/server001.key'
          ssl_password: 'qwerty'
          ssl_ciphers: 'ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256'
  instance002:
    iproto:
      listen:
      - uri: '127.0.0.1:3302'
        params:
          transport: 'ssl'
          ssl_ca_file: 'certs/root_ca.crt'
          ssl_cert_file: 'certs/instance002/server002.crt'
          ssl_key_file: 'certs/instance002/server002.key'
          ssl_password_file: 'certs/ssl_passwords.txt'
          ssl_ciphers: 'ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256'
  instance003:
    iproto:
      listen:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      - uri: '127.0.0.1:3303'
        params:
          transport: 'ssl'
          ssl_ca_file: 'certs/root_ca.crt'
          ssl_cert_file: 'certs/instance003/server003.crt'
          ssl_key_file: 'certs/instance003/server003.key'
          ssl_password_file: 'certs/ssl_passwords.txt'
          ssl_ciphers: 'ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256'
```

You can find the full example here: ssl_with_ca.

### Advertise URI specifics

SSL parameters for an advertise URI should be set only if this *advertise URI is specified explicitly*. Otherwise, SSL parameters of a listen URI are used and no additional configuration is required.

Configuring an advertise URI's SSL options depends on whether a trusted certificate authorities (CA) file is set or not. Without the CA file, you only need to set `iproto.advertise.<peer_or_sharding>.params.transport` to `ssl` as shown below:

```
instance001:
  iproto:
    listen:
    - uri: '192.168.0.101:3301'
      params:
        transport: 'ssl'
        ssl_cert_file: 'certs/server.crt'
        ssl_key_file: 'certs/server.key'
    advertise:
      peer:
        uri: 'server.example.com:3301'
        params:
          transport: 'ssl'
```

If the CA file is specified for a listen URI, you also need to configure `ssl_cert_file` and `ssl_key_file` for this advertise URI:

```
instance001:
  iproto:
    listen:
    - uri: '192.168.0.101:3301'
      params:
        transport: 'ssl'
        ssl_ca_file: 'certs/root_ca.crt'
        ssl_cert_file: 'certs/instance001/server001.crt'
        ssl_key_file: 'certs/instance001/server001.key'
    advertise:
      peer:
        uri: 'server001.example.com:3301'
        params:
          transport: 'ssl'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        ssl_cert_file: 'certs/instance001/server001.crt'
        ssl_key_file: 'certs/instance001/server001.key'
```

### Reloading certificates

To reload SSL certificate files specified in the configuration, open an *admin console* and reload the configuration using *config.reload()*:

```
require('config'):reload()
```

New certificates will be used for new connections. Existing connections will continue using old SSL certificates until reconnection is required. For example, certificate expiry or a network issue causes reconnection.

## Credentials

Tarantool enables flexible management of access to various database resources by providing specific privileges to users. You can read more about the main concepts of Tarantool access control system in the *Управление доступом* section.

This topic describes how to create users and grant them the specified privileges in the *credentials* section of a YAML configuration. For example, you can define users with the `replication` and `sharding` roles to maintain *replication* and sharding in a Tarantool cluster.

### Managing users and roles

#### Creating a user

You can create new or configure credentials of the existing users in the *credentials.users* section.

In the example below, a `dbadmin` user without a password is created:

```
credentials:
  users:
    dbadmin: {}
```

To set a password, use the *credentials.users.<username>.password* option:

```
credentials:
  users:
    dbadmin:
      password: 'T0p_Secret_P@$$w0rd'
```

### Granting privileges to a user

To assign a role to a user, use the *credentials.users.<username>.roles* option. In this example, the `dbadmin` user gets privileges granted to the `super` built-in role:

```
credentials:
  users:
    dbadmin:
      password: 'T0p_Secret_P@$$w0rd'
      roles: [ super ]
```

To create a new role, define it in the *credentials.roles.** section. In the example below, the `writers_space_reader` role gets privileges to select data in the `writers` space:

```
roles:
  writers_space_reader:
    privileges:
    - permissions: [ read ]
      spaces: [ writers ]
```

Then, you can assign this role to a user using *credentials.users.<username>.roles* (`sampleuser` in the example below):

```
sampleuser:
  password: '123456'
  roles: [ writers_space_reader ]
```

You can grant specific privileges directly using *credentials.users.<username>.privileges*. In this example, `sampleuser` gets privileges to select and modify data in the `books` space:

```
sampleuser:
  password: '123456'
  roles: [ writers_space_reader ]
  privileges:
  - permissions: [ read, write ]
    spaces: [ books ]
```

You can find the full example here: credentials.

### Loading secrets from safe storage

Tarantool enables you to load secrets from safe storage such as external files or environment variables. To do this, you need to define corresponding options in the *config.context* section. In the examples below, `context.dbadmin_password` and `context.sampleuser_password` define how to load user passwords from `*.txt` files or environment variables:

- This example shows how to load passwords from `*.txt` files:

```
config:
  context:
    dbadmin_password:
      from: file
      file: secrets/dbadmin_password.txt
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      rstrip: true
    sampleuser_password:
      from: file
      file: secrets/sampleuser_password.txt
      rstrip: true
```

- This example shows how to load passwords from environment variables:

```
config:
  context:
    dbadmin_password:
      from: env
      env: DBADMIN_PASSWORD
    sampleuser_password:
      from: env
      env: SAMPLEUSER_PASSWORD
```

These environment variables should be set before *starting instances*.

After configuring how to load passwords, you can set password values using *credentials.users.<username>.password* as follows:

```
credentials:
  users:
    dbadmin:
      password: '{{ context.dbadmin_password }}'
    sampleuser:
      password: '{{ context.sampleuser_password }}'
```

You can find the full examples here: credentials_context_file, credentials_context_env.

### Authentication

**Enterprise Edition**

Authentication features are supported by the Enterprise Edition only.

### Authentication restrictions

Tarantool Enterprise Edition provides the ability to apply additional restrictions for user authentication. For example, you can specify the minimum time between authentication attempts or turn off access for guest users.

In the configuration below, *security.auth_retries* is set to 2, which means that Tarantool lets a client try to authenticate with the same username three times. At the fourth attempt, the authentication delay configured with *security.auth_delay* is enforced. This means that a client should wait 10 seconds after the first failed attempt.

```
security:
  auth_delay: 10
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  auth_retries: 2
  disable_guest: true
```

The *disable_guest* option turns off access over remote connections from unauthenticated or *guest* users.

### Password policy

A password policy allows you to improve database security by enforcing the use of strong passwords, setting up a maximum password age, and so on. When you create a new user with *box.schema.user.create* or update the password of an existing user with *box.schema.user.passwd*, the password is checked against the configured password policy settings.

In the example below, the following options are specified:

- *password_min_length* specifies that a password should be at least 16 characters.
- *password_enforce_lowercase* and *password_enforce_uppercase* specify that a password should contain lowercase and uppercase letters.
- *password_enforce_digits* and *password_enforce_specialchars* specify that a password should contain digits and at least one special character.
- *password_lifetime_days* sets a maximum password age to 365 days.
- *password_history_length* specifies that a new password should differ from the last three passwords.

```
security:
  password_min_length: 16
  password_enforce_lowercase: true
  password_enforce_uppercase: true
  password_enforce_digits: true
  password_enforce_specialchars: true
  password_lifetime_days: 365
  password_history_length: 3
```

### Authentication protocol

By default, Tarantool uses the CHAP protocol to authenticate users and applies `SHA-1` hashing to *passwords*. Note that CHAP stores password hashes in the `_user` space unsalted. If an attacker gains access to the database, they may crack a password, for example, using a rainbow table.

In the Enterprise Edition, you can enable PAP authentication with the `SHA256` hashing algorithm. For PAP, a password is salted with a user-unique salt before saving it in the database, which keeps the database protected from cracking using a rainbow table.

To enable PAP, specify the *security.auth_type* option as follows:

```
security:
  auth_type: 'pap-sha256'
```

For new users, the *box.schema.user.create* method generates authentication data using `PAP-SHA256`. For existing users, you need to reset a password using *box.schema.user.passwd* to use the new authentication protocol.

**Предупреждение:** Given that `PAP` transmits a password as plain text, Tarantool requires configuring *SSL/TLS* for a connection.

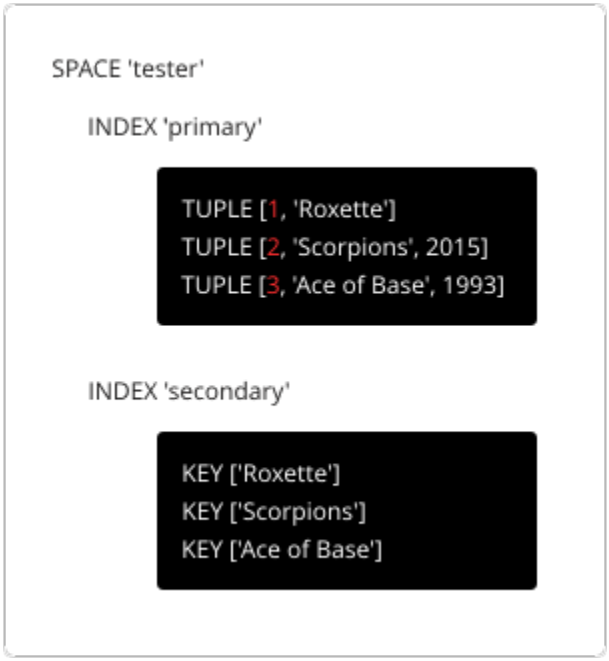
The example below shows how to specify the authentication protocol using the `auth_type` parameter when connecting to an instance using *net.box*:

```
local connection = require('net.box').connect({
    uri = 'admin:topsecret@127.0.0.1:3301',
    params = { auth_type = 'pap-sha256',
               transport = 'ssl',
               ssl_cert_file = 'certs/server.crt',
               ssl_key_file = 'certs/server.key' }
})
```

If the authentication protocol isn't specified explicitly on the client side, the client uses the protocol configured on the server via `security.auth_type`.

### 4.8.2 Модель данных

Если вы пробовали создать базу данных, как предлагается в упражнениях в *базовом практическом руководстве*, то ваша тестовая база данных выглядит следующим образом:




Tarantool stores data in *spaces*, which can be thought of as tables in a relational database. Every record or row in a space is called a *tuple*. A tuple may have any number of fields, and the fields may be of different *types*.

String data in fields are compared based on the specified *collation* rules. The user can provide hard limits for data values through *constraints* and link related spaces with *foreign keys*.

Tarantool supports highly customizable *indexes* of various types. In particular, indexes can be defined with generators like *sequences*.

There are six basic *data operations* in Tarantool: SELECT, INSERT, UPDATE, UPSERT, REPLACE, and DELETE. A number of *complexity factors* affects the resource usage of each function.

Tarantool allows *describing the data schema* but does not require it. The user can *migrate a schema* without migrating the data.

To ensure *data persistence* and recover quickly in case of failure, Tarantool uses mechanisms like the write-ahead log (WAL) and snapshots.

### Хранение данных

### Кортежи

Tarantool обрабатывает данные в виде кортежей.

**кортеж** Кортеж — это группа значений данных в памяти Tarantool. По сути, это «запись в базе данных» или «строка». Значения данных в кортеже называются *полями*.

Когда Tarantool выводит значение кортежа в консоль, по умолчанию используется формат *YAML*, например: `[3, 'Ace of Base', 1993]`.

В Tarantool кортежи хранятся в виде массивов в формате MsgPack.

**поле** Поля — это отдельные значения данных, которые содержатся в кортеже. Они играют ту же роль, что и «столбцы» или «поля записи» в реляционных базах данных, но несколько усовершенствованы:

- поля могут представлять собой композитные структуры, такие как таблицы типа массива или ассоциативного массива,
- полям не нужны имена.

В кортеже может быть любое количество полей, и это могут быть поля разных *типов*.

Номер поля служит его идентификатором. В Lua и некоторых других языках нумерация начинается с 1, в других — с 0 (например, в PHP или C/C++). Таким образом, в некоторых контекстах у первого поля кортежа будет индекс `1` или `0`.

### Спейсы

Tarantool stores tuples in containers called spaces.

**спейс** В Tarantool спейс — это первичный контейнер, хранящий данные. Он похож на таблицы в реляционных базах данных. Спейсы содержат *кортежи* — так в Tarantool называются записи в базе данных. Количество кортежей в спейсе не ограничено.

Для хранения данных с помощью Tarantool требуется хотя бы один спейс. У каждого спейса есть следующие атрибуты:

- уникальное **имя**, указанное пользователем;
- уникальный **числовой идентификатор**, обычно Tarantool назначает его автоматически, но пользователь может его указать сам, если посчитает нужным;
- **движок**: *memtx* (по умолчанию) — движок «in-memory», быстрый, но ограниченный в размере, или *vinyl* — дисковый движок для огромных наборов данных.

Для работы спейсу нужен *первичный индекс*. Также он может использовать вторичные индексы.

### Типы данных

Tarantool представляет собой систему управления базой данных и сервер приложений одновременно. Поэтому разработчику часто приходится работать с двумя системами типов данных: типы языка программирования (например, Lua) и типы формата хранения данных Tarantool (MsgPack).

### Lua в сравнении с MsgPack

| Скалярный / составной | MsgPack type | Lua-тип | Пример значения |
|---|---|---|---|
| скалярный | nil | cdata | *box.NULL* |
| скалярный | boolean | boolean | `true` |
| скалярный | string | string | `'A B C'` |
| скалярный | integer | number | `12345` |
| скалярный | integer | cdata | `12345` |
| скалярный | float64 (double) | number | `1.2345` |
| скалярный | float64 (double) | cdata | `1.2345` |
| скалярный | binary | cdata | `[!!binary 3t7e]` |
| скалярный | ext (for Tarantool `decimal`) | cdata | `1.2` |
| скалярный | ext (for Tarantool `datetime`) | cdata | `'2021-08-20T16:21:25.122999906 Europe/Berlin'` |
| скалярный | ext (for Tarantool `interval`) | cdata | `+1 months, 1 days` |
| скалярный | ext (for Tarantool `uuid`) | cdata | `12a34b5c-de67-8f90-123g-h4567ab8901` |
| составной | map (ассоциативный массив) | table (with string keys) | `{'a': 5, 'b': 6}` |
| составной | array (массив) | table (with integer keys) | `[1, 2, 3, 4, 5]` |
| составной | array (массив) | tuple (cdata) | `[12345, 'A B C']` |

**Примечание:** Данные в формате MsgPack имеют переменный размер. Так, например, для наименьшего значения number потребуется только один байт, а для наибольшего потребуется девять байтов.

**Примечание:** The Lua nil type is encoded as MsgPack `nil` but decoded as *msgpack.NULL*.

### Описание типов полей

#### nil

In Lua, the nil type has only one possible value, also called `nil`. Tarantool displays it as `null` when using the default *YAML* format. Nil may be compared to values of any types with == (is-equal) or ~= (is-not-equal), but other comparison operations will not work. Nil may not be used in Lua tables; the workaround is to use *box.NULL* because `nil == box.NULL` is true. **Example:** `nil`.

### boolean

A boolean is either `true` or `false`.

**Example:** `true`.

### integer

The Tarantool integer type is for integers between -9223372036854775808 and 18446744073709551615, which is about 18 quintillion. This type corresponds to the number type in Lua and to the integer type in MsgPack.

**Example:** `-2^63`.

### unsigned

The Tarantool unsigned type is for integers between 0 and 18446744073709551615. So it is a subset of integer.

**Example:** `123456`.

### double

The double field type exists mainly to be equivalent to Tarantool/SQL's *DOUBLE data type*. In msgpuck.h (Tarantool's interface to MsgPack), the storage type is `MP_DOUBLE` and the size of the encoded value is always 9 bytes. In Lua, fields of the double type can only contain non-integer numeric values and cdata values with double floating-point numbers.

**Examples:** `1.234`, `-44`, `1.447e+44`.

To avoid using the wrong kind of values inadvertently, use `ffi.cast()` when searching or changing `double` fields. For example, instead of *space_object*`:insert{`*value*`}` use `ffi = require('ffi') ...` *space_object*`:insert({ffi.cast('double',`*value*`)})`.

**Example:**

```
s = box.schema.space.create('s', {format = {{'d', 'double'}}})
s:create_index('ii')
s:insert({1.1})
ffi = require('ffi')
s:insert({ffi.cast('double', 1)})
s:insert({ffi.cast('double', tonumber('123'))})
s:select(1.1)
s:select({ffi.cast('double', 1)})
```

Арифметические операции с cdata формата `double` работают ненадёжно, поэтому для Lua лучше использовать тип `number`. Это не относится к Tarantool/SQL, так как Tarantool/SQL применяет *неявное приведение типов*.

#### number

The Tarantool number field may have both integer and floating-point values, although in Lua a `number` is a double-precision floating-point.

Tarantool по возможности сохраняет числа языка Lua в виде чисел с плавающей запятой, если числовое значение содержит десятичную запятую или если оно очень велико (более 100 триллионов = 1e14). В противном случае Tarantool сохраняет такое значение в виде целого числа. Чтобы даже очень большие величины гарантированно сохранялись как целые числа, используйте функцию *tonumber64* или приписывайте в конце суффикс LL (Long Long) или ULL (Unsigned Long Long). Вот примеры записи чисел в обычном представлении, экспоненциальном, с суффиксом ULL и с использованием функции `tonumber64`: `55`, `2.7e+20`, `100000000000000ULL`, `tonumber64('18446744073709551615')`.

You can also use the `ffi` module to specify a C type to cast the number to. In this case, the number will be stored as cdata.

#### decimal

The Tarantool decimal type is stored as a *MsgPack ext* (Extension). Values with the decimal type are not floating-point values although they may contain decimal points. They are exact with up to 38 digits of precision.

**Example:** a value returned by a function in the *decimal* module.

#### datetime

Introduced in v. 2.10.0. The Tarantool `datetime` type facilitates operations with date and time, accounting for leap years or the varying number of days in a month. It is stored as a *MsgPack ext* (Extension). Operations with this data type use code from c-dt, a third-party library.

For more information, see *Module datetime*.

#### interval

**Since:** v. 2.10.0

The Tarantool `interval` type represents periods of time. They can be added to or subtracted from `datetime` values or each other. Operations with this data type use code from c-dt, a third-party library. The type is stored as a *MsgPack ext* (Extension). For more information, see *Module datetime*.

#### string

A string is a variable-length sequence of bytes, usually represented with alphanumeric characters inside single quotes. In both Lua and MsgPack, strings are treated as binary data, with no attempts to determine a string's character set or to perform any string conversion – unless there is an optional *collation*. So, usually, string sorting and comparison are done byte-by-byte, without any special collation rules applied. For example, numbers are ordered by their point on the number line, so 2345 is greater than 500; meanwhile, strings are ordered by the encoding of the first byte, then the encoding of the second byte, and so on, so `'2345'` is less than `'500'`.

**Example:** `'A, B, C'`.

#### bin

A bin (binary) value is not directly supported by Lua but there is a Tarantool type `varbinary` which is encoded as MsgPack binary. For an (advanced) example showing how to insert varbinary into a database, see the Cookbook Recipe for *ffi_varbinary_insert*.

**Example:** `"\65 \66 \67"`.

#### uuid

The Tarantool uuid type is used for *Universally Unique Identifiers*. Since version *2.4.1* Tarantool stores `uuid` values as a *MsgPack ext* (Extension).

**Example:** `64d22e4d-ac92-4a23-899a-e5934af5479`.

#### array (массив)

An array is represented in Lua with `{...}` (braces).

**Examples:** lists of numbers representing points in geometric figures: `{10, 11}`, `{3, 5, 9, 10}`.

#### table

Lua tables with string keys are stored as MsgPack maps; Lua tables with integer keys starting with 1 are stored as MsgPack arrays. Nils may not be used in Lua tables; the workaround is to use *box.NULL*.

**Example:** a `box.space.tester:select()` request will return a Lua table.

#### кортеж

A tuple is a light reference to a MsgPack array stored in the database. It is a special type (cdata) to avoid conversion to a Lua table on retrieval. A few functions may return tables with multiple tuples. For tuple examples, see *box.tuple*.

#### скалярный

Values in a scalar field can be boolean, integer, unsigned, double, number, decimal, string, uuid, or varbinary; but not array, map, or tuple.

**Examples:** `true`, `1`, `'xxx'`.

#### any

Values in a field of this type can be boolean, integer, unsigned, double, number, decimal, string, uuid, varbinary, array, map, or tuple.

**Examples:** `true`, `1`, `'xxx'`, `{box.NULL, 0}`.

### Examples

Примеры запросов вставки с разными типами полей:

```
tarantool> box.space.K:insert{1,nil,true,'A B C',12345,1.2345}
---
- [1, null, true, 'A B C', 12345, 1.2345]
...
tarantool> box.space.K:insert{2,{['a']=5,['b']=6}}
---
- [2, {'a': 5, 'b': 6}]
...
tarantool> box.space.K:insert{3,{1,2,3,4,5}}
---
- [3, [1, 2, 3, 4, 5]]
...
```

### Типы индексированных полей

О том, какие значения можно хранить в индексированных полях, читайте в разделе *об индексах*»

### Сортировка

Когда Tarantool сравнивает строки, по умолчанию он использует **двоичные параметры сортировки** (**binary collation**). При этом он учитывает только числовое значение каждого байта в строке. Например, код символа `'A'` (раньше называлась «значение ASCII») — число 65, код `'B'` — число 66, а код `'a'` – число 98. Поэтому `'A' < 'B' < 'a'`, если строка закодирована в ASCII или UTF-8.

Двоичная сортировка — лучший выбор для быстрого детерминированного простого обслуживания и поиска с использованием индексов Tarantool.

Но если вы хотите такое упорядочение, как в телефонных справочниках и словарях, то вам нужна одна из дополнительных сортировок Tarantool: `unicode` или `unicode_ci`. Они обеспечивают `'a' < 'A' < 'B'` и `'a' == 'A' < 'B'` соответственно.

Дополнительные виды сортировки **unicode и unicode_ci** обеспечивают упорядочение в соответствии с Таблицей сортировки символов Юникода по умолчанию (DUCET) и правилами, указанными в Техническом стандарте Юникода №10 – Алгоритм сортировки по Юникоду (Unicode® Technical Standard #10 Unicode Collation Algorithm (UTS #10 UCA)). Единственное отличие между двумя видами сортировки — вес:

- сортировка `unicode` принимает во внимание уровни веса L1, L2 и L3 (уровень = „tertiary“, третичный);
- сортировка `unicode_ci` принимает во внимание только вес L1 (уровень = „primary“, первичный), поэтому, например, `'a' == 'A' == 'á' == 'Á'`.

Для примера возьмем некоторые русские слова:

```
'ЕЛЕ'
'елейный'
'ёлка'
'еловый'
'елозить'
'Ёлочка'
'ёлочный'
'ЕЛь'
'ель'
```

. . . и покажем разницу в упорядочении и выборке по индексу:

- с сортировкой по `unicode`:

```
tarantool> box.space.T:create_index('I', {parts = {{field = 1, type = 'str',␣
↪collation='unicode'}}})
...
tarantool> box.space.T.index.I:select()
---
- - ['ЕЛЕ']
  - ['елейный']
  - ['ёлка']
  - ['еловый']
  - ['елозить']
  - ['Ёлочка']
  - ['ёлочный']
  - ['ель']
  - ['ЕЛь']
...
tarantool> box.space.T.index.I:select{'ЁлКа'}
---
- []
...
```

- с сортировкой по `unicode_ci`:

```
tarantool> box.space.T:create_index('I', {parts = {{field = 1, type ='str',␣
↪collation='unicode_ci'}}})
...
tarantool> box.space.T.index.I:select()
---
- - ['ЕЛЕ']
  - ['елейный']
  - ['ёлка']
  - ['еловый']
  - ['елозить']
  - ['Ёлочка']
  - ['ёлочный']
  - ['ЕЛь']
...
tarantool> box.space.T.index.I:select{'ЁлКа'}
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- - ['ёлка']
...
```

Сортировка включает в себя ещё множество аспектов, кроме показанного в этом примере сопоставления букв верхнего и нижнего регистра, с диакритическими знаками и без них, и с учётом алфавита. Учитываются также вариации одного и того же символа, системы неалфавитного письма и применяются специальные правила для комбинаций символов.

Для английского, русского и большинства других языков используйте «unicode» и «unicode_ci». Если вам нужно, чтобы у кириллических букв „Е“ и „Ё“ веса 1 уровня были одинаковыми, попробуйте киргизскую сортировку.

**Специализированные дополнительные виды сортировки**: Для других языков Tarantool предлагает специализированные виды сортировки для любого современного языка, на котором говорят более миллиона человек. Кроме того, специализированные дополнительные виды сортировки возможны для особых случаев, когда слова в словаре упорядочиваются не так, как в телефонном справочнике. Чтобы увидеть полный список, выполните команду `box.space._collation:select()`.

Названия специализированных видов сортировки имеют вид `unicode_[language code]_[strength]`, где language code — это стандартный код языка из 2 или 3 символов, а значение strength может быть `s1` для уровня «primary» (вес уровня 1), `s2` для уровня «secondary», `s3` для уровня «tertiary». Tarantool использует те же коды языков, что указаны в списке специализированных вариантов языковых настроек на страницах руководств по Ubuntu и Fedora. Схемы, в деталях объясняющие отличия от упорядочения по DUCET, можно найти в Общем репозитории языковых данных (Common Language Data Repository).

### Constraints

For better control over stored data, Tarantool supports **constraints** – user-defined limitations on the values of certain fields or entire tuples. Together with data types, constraints allow limiting the ranges of available field values both syntactically and semantically.

For example, the field `age` typically has the `number` type, so it cannot store strings or boolean values. However, it can still have values that don't make sense, such as negative numbers. This is where constraints come to help.

### Constraint types

There are two types of constraints in Tarantool:

- *Field constraints* check that the value being assigned to a field satisfies a given condition. For example, `age` must be non-negative.
- *Tuple constraints* check complex conditions that can involve all fields of a tuple. For example, a tuple contains a date in three fields: `year`, `month`, and `day`. You can validate `day` values based on the `month` value (and even `year` if you consider leap years).

Field constraints work faster, while tuple constraints allow implementing a wider range of limitations.

### Constraint functions

Constraints use stored Lua functions or *SQL expressions*, which must return `true` when the constraint is satisfied. Other return values (including `nil`) and exceptions make the check fail and prevent tuple insertion or modification.

To create a constraint function, call *box.schema.func.create()* with the function definition specified in the `body` attribute.

Constraint functions take two parameters:

- The tuple and the constraint name for tuple constraints.

```
-- Define a tuple constraint function --
box.schema.func.create('check_person', {
    language = 'LUA',
    is_deterministic = true,
    body = 'function(t, c) return (t.age >= 0 and #(t.name) > 3) end'
})
```

> **Предупреждение:** Tarantool doesn't check field names used in tuple constraint functions. If a field referenced in a tuple constraint gets renamed, this constraint will break and prevent further insertions and modifications in the space.

- The field value and the constraint name for field constraints.

```
-- Define a field constraint function --
box.schema.func.create('check_age', {
    language = 'LUA',
    is_deterministic = true,
    body = 'function(f, c) return (f >= 0 and f < 150) end'
})
```

### Creating constraints

To create a constraint in a space, specify the corresponding function's name in the `constraint` parameter:

- Tuple constraints: when creating or altering a space.

```
-- Create a space with a tuple constraint --
customers = box.schema.space.create('customers', {constraint = 'check_person'})
```

- Field constraints: when setting up the space format.

```
-- Specify format with a field constraint --
box.space.customers:format({
    {name = 'id', type = 'number'},
    {name = 'name', type = 'string'},
    {name = 'age',  type = 'number', constraint = 'check_age'},
})
```

In both cases, `constraint` can contain multiple function names passed as a tuple. Each constraint can have an optional name:

```
-- Create one more tuple constraint --
box.schema.func.create('another_constraint',
    {language = 'LUA', is_deterministic = true, body = 'function(t, c) return true end'})

-- Set two constraints with optional names --
box.space.customers:alter{
    constraint = { check1 = 'check_person', check2 = 'another_constraint'}
}
```

**Примечание:** When adding a constraint to an existing space with data, Tarantool checks it against the stored data. If there are fields or tuples that don't satisfy the constraint, it won't be applied to the space.

### Foreign keys

**Foreign keys** provide links between related fields, therefore maintaining the referential integrity of the database.

Fields can contain values that exist only in other fields. For example, a shop order always belongs to a customer. Hence, all values of the `customer` field of the `orders` space must also exist in the `id` field of the `customers` space. In this case, `customers` is a **parent space** for `orders` (its **child space**). When two spaces are linked with a foreign key, each time a tuple is inserted or modified in the child space, Tarantool checks that a corresponding value is present in the parent space.

**Примечание:** A foreign key can link a field to another field in the same space. In this case, the child field must be nullable. Otherwise, it is impossible to insert the first tuple in such a space because there is no parent tuple to which it can link.

### Foreign key types

There are two types of foreign keys in Tarantool:

- *Field foreign keys* check that the value being assigned to a field is present in a particular field of another space. For example, the `customer` value in a tuple from the `orders` space must match an `id` stored in the `customers` space.
- *Tuple foreign keys* check that multiple fields of a tuple have a match in another space. For example, if the `orders` space has fields `customer_id` and `customer_name`, a tuple foreign key can check that the `customers` space contains a tuple with both these values in the corresponding fields.

Field foreign keys work faster while tuple foreign keys allow implementing more strict references.

### Creating foreign keys

**Важно:** For each foreign key, there must exist a parent space index that includes all its fields.

To create a foreign key in a space, specify the parent space and linked fields in the `foreign_key` parameter. Parent spaces can be referenced by name or by id. When linking to the same space, the space can be omitted. Fields can be referenced by name or by number:

- Field foreign keys: when setting up the space format.

```
-- Create a space with a field foreign key --
box.schema.space.create('orders')

box.space.orders:format({
    {name = 'id',   type = 'number'},
    {name = 'customer_id', foreign_key = {space = 'customers', field = 'id'}},
    {name = 'price_total', type = 'number'},
})
```

- Tuple foreign keys: when creating or altering a space. Note that for foreign keys with multiple fields there must exist an index that includes all these fields.

```
-- Create a space with a tuple foreign key --
box.schema.space.create("orders", {
    foreign_key = {
        space = 'customers',
        field = {customer_id = 'id', customer_name = 'name'}
    }
})

box.space.orders:format({
    {name = "id", type = "number"},
    {name = "customer_id" },
    {name = "customer_name"},
    {name = "price_total", type = "number"},
})
```

**Примечание:** Type can be omitted for foreign key fields because it's defined in the parent space.

Foreign keys can have an optional name.

```
-- Set a foreign key with an optional name --
box.space.orders:alter{
    foreign_key = {
        customer = {
            space = 'customers',
            field = { customer_id = 'id', customer_name = 'name'}
        }
    }
}
```

A space can have multiple tuple foreign keys. In this case, they all must have names.

```
-- Set two foreign keys: names are mandatory --
box.space.orders:alter{
    foreign_key = {
        customer = {
            space = 'customers',
            field = {customer_id = 'id', customer_name = 'name'}
        },
        item = {
            space = 'items',
            field = {item_id = 'id'}
        }
    }
}
```

Tarantool performs integrity checks upon data modifications in parent spaces. If you try to remove a tuple referenced by a foreign key or an entire parent space, you will get an error.

---

**Важно:** Renaming parent spaces or referenced fields may break the corresponding foreign keys and prevent further insertions or modifications in the child spaces.

---

### Индексы

### Basics

**Индекс** — это специальная структура данных, которая хранит группу ключевых значений и указателей. Индекс используется для эффективного управления данными.

Как и для спейсов, для индексов следует указать **имена**, а Tarantool определит уникальный **числовой идентификатор** («ID индекса»).

У индекса всегда есть определенный **тип**. Тип индекса по умолчанию — *TREE*. TREE-индексы поддерживаются обоими движками Tarantool, могут индексировать уникальные и неуникальные значения, поддерживают поиск по компонентам ключа, сравнение ключей и упорядочивание результатов. Движок memtx поддерживает и другие типы индексов: *HASH*, *RTREE* и *BITSET*.

Индекс может быть **многокомпонентным**, то есть можно объявить, что ключ индекса состоит из двух или более полей в кортеже в любом порядке. Например, для обычного TREE-индекса максимальное количество частей равно 255.

Индекс может быть **уникальным**, то есть можно объявить, что недопустимо дважды задавать одно значение ключа.

Первый индекс, определенный для спейса, называется **первичный индекс**. Он должен быть уникальным. Все остальные индексы называются **вторичными индексами**, они могут строиться по неуникальным значениям.

На индексы распространяются определенные ограничения. Более подробную информацию см. на странице *Ограничения*.

Можно создать генератор индексов, используя последовательность (sequence). Чтобы узнать подробности, обратитесь к *практическому руководству*.

### Типы индексированных полей

Типы полей не следует путать с *типами индексов* как структур данных. О типах индексов можно прочитать *ниже*.

Индексы ограничивают значения, которые Tarantool может хранить в формате MsgPack. Вот почему, например, есть отдельные типы полей `'unsigned'` (число без знака) и `'integer'` (целое число), хотя в MsgPack они оба хранятся как целочисленные значения. Индекс типа `'unsigned'` содержит только *неотрицательные* целочисленные значения, а индекс типа `'integer'` содержит *любые* целочисленные значения.

The default field type is `'unsigned'` and the default index type is TREE. Although `'nil'` is not a legal indexed field type, indexes may contain `nil` *as a non-default option*.

To learn more about field types, check the *Field type details* section.

| Имя типа поля | Field type | Тип индекса |
|---|---|---|
| `'boolean'` | *boolean* | *boolean* |
| `'integer'` (may also be called `'int'`) | *integer*, может включать в себя значения unsigned (без знака) | TREE или HASH |
| `'unsigned'` (без знака, также может называться `'uint'` или `'num'`, но `'num'` объявлен устаревшим) | *unsigned* | TREE, BITSET или HASH |
| `'double'` | *double* | TREE или HASH |
| `'number'` | *number*, может включать в себя значения типа *integer*, *double* или *decimal* | TREE или HASH |
| `'decimal'` | *decimal* | TREE или HASH |
| `'string'` (строка, также может называться `'str'`) | *string* | TREE, BITSET или HASH |
| `'varbinary'` | *varbinary* | TREE, HASH, or BITSET (since version *2.7.1*) |
| `'uuid'` | *uuid* | TREE или HASH |
| `datetime` | *datetime* | TREE |
| `'array'` | *array* | *RTREE* |
| `'scalar'` | may include *nil*, *boolean*, *integer*, *unsigned*, *number*, *decimal*, *string*, *varbinary*, or *uuid* values \| When a scalar field contains values of different underlying types, the key order is: nils, then booleans, then numbers, then strings, then varbinaries, then uuids. | TREE или HASH |

### Типы индексов

Индекс всегда относится к определенному **типу**. Для разных сценариев использования требуются разные типы индексов.

Обзор характеристик индексов приведен в следующей таблице:

| Характеристика | TREE | HASH | RTREE | BITSET |
|---|---|---|---|---|
| уникальный | + | + | - | - |
| неуникальный | + | - | + | + |
| *is_nullable* | + | - | - | - |
| может быть составным (multipart) | + | + | - | - |
| *индекс по массиву (multikey)* | + | - | - | - |
| *поиск по части ключа* | + | - | - | - |
| может быть по первичному ключу | + | + | - | - |
| `exclude_null` (версии 2.8+) | + | - | - | - |
| Pagination (the *after* option) | + | - | - | - |
| *типы итераторов* | ALL, EQ, REQ, GT, GE, LT, LE | ALL, EQ | ALL, EQ, GT, GE, LT, LE, OVERLAPS, NEIGHBOR | ALL, EQ, BITS_ALL_SET, BITS_ANY_SET, BITS_ALL_NOT_SET |

**Примечание:** In *2.11.0*, the `GT` index type is deprecated for HASH indexes.

### TREE-индексы

Тип индекса по умолчанию — „TREE“. Движки memtx и vinyl предоставляют TREE-индексы, которые могут индексировать уникальные и неуникальные значения, поддерживают поиск по компонентам ключа, сравнение ключей и упорядоченные результаты.

Это универсальный тип индексов, в большинстве случаев он подойдет лучше всего.

Кроме того, движок memtx поддерживает следующие типы индексов: HASH, RTREE и BITSET.

### HASH-индексы

HASH-индексы требуют уникальности полей и почти во всех отношениях проигрывают индексам типа TREE. Поэтому мы не рекомендуем использовать их в приложениях. В настоящее время Tarantool поддерживает HASH в основном для обратной совместимости.

Вот несколько советов. Не используйте HASH-индекс:

- если просто хочется
- если вы думаете, что HASH быстрее, без измерения производительности
- если вы хотите выполнять перебор данных
- по первичному ключу
- как единственный индекс

Используйте HASH-индекс:

- если это вторичный ключ
- если вы на сто процентов уверены, что его не придется делать неуникальным
- если вы обнаружили значимое увеличение производительности в ходе проведенных измерений
- если вы экономите каждый байт на кортежах (HASH немного компактнее)

### RTREE-индексы

RTREE — это многомерный индекс, который поддерживает до 20 измерений. Он используется, в частности, для индексирования пространственной информации, такой как географические объекты. В *этом примере* мы показываем, как использовать пространственный поиск с помощью RTREE-индекса.

RTREE-индекс не может быть первичным и не может быть уникальным. Индекс такого типа может содержать параметры измерения и расстояния — `dimension` и `distance` соответственно. Определение `parts` должно включать только один компонент типа `array`. RTREE-индекс может принимать два типа функции расстояния `distance`: `euclid`, то есть Евклидова метрика, или `manhattan`, то есть расстояние городских кварталов.

**Предупреждение:** Currently, the *isolation level* of RTREE indexes in *MVCC transaction mode* is *read-committed* (not *serializable*, as stated). If a transaction uses these indexes, it can read committed or confirmed data (depending on the isolation level). However, the indexes are subject to different anomalies that can make them unserializable.

**Пример 1:**

```
my_space = box.schema.create_space("tester")
my_space:format({ { type = 'number', name = 'id' }, { type = 'array', name = 'content' }␣
↪})
hash_index = my_space:create_index('primary', { type = 'tree', parts = {'id'} })
rtree_index = my_space:create_index('spatial', { type = 'RTREE', unique = false, parts =
↪{'content'} })
```

Таким образом, соответствующее поле кортежа может быть массивом типа array из 2 или 4 чисел. 2 числа обозначают точку {x, y}; 4 числа обозначают прямоугольник {x1, y1, x2, y2}, где (x1, y1) и (x2, y2) — диагональные точки прямоугольника.

```
my_space:insert{1, {1, 1}}
my_space:insert{2, {2, 2, 3, 3}}
```

Результаты выборки зависят от выбранного итератора. Итератор EQ, который используется по умолчанию, ищет точный прямоугольник, точка рассматривается как прямоугольник нулевой ширины и высоты:

```
tarantool> rtree_index:select{1, 1}
---
- - [1, [1, 1]]
...

tarantool> rtree_index:select{1, 1, 1, 1}
---
- - [1, [1, 1]]
...

tarantool> rtree_index:select{2, 2}
---
- []
...

tarantool> rtree_index:select{2, 2, 3, 3}
---
- - [2, [2, 2, 3, 3]]
...
```

Итератор ALL, который используется по умолчанию, если не указан ключ, выбирает все кортежи в произвольном порядке:

```
tarantool> rtree_index:select{}
---
- - [1, [1, 1]]
  - [2, [2, 2, 3, 3]]
...
```

Итератор LE (меньше или равно) ищет кортежи с прямоугольниками, которые находятся в пределах заданного прямоугольника:

```
tarantool> rtree_index:select({1, 1, 2, 2}, {iterator='le'})
---
- - [1, [1, 1]]
...
```

Итератор LT (меньше или строго меньше) ищет кортежи с прямоугольниками, которые находятся строго в пределах заданного прямоугольника:

```
tarantool> rtree_index:select({0, 0, 3, 3}, {iterator = 'lt'})
---
- - [1, [1, 1]]
...
```

Итератор GE ищет кортежи с прямоугольниками, в пределах которых находится заданный прямоугольник:

```
tarantool> rtree_index:select({1, 1}, {iterator = 'ge'})
---
- - [1, [1, 1]]
...
```

Итератор GT ищет кортежи с прямоугольниками, строго в пределах которых находится заданный прямоугольник:

```
tarantool> rtree_index:select({2.1, 2.1, 2.9, 2.9}, {iterator = 'gt'})
---
- []
...
```

Итератор OVERLAPS ищет кортежи с прямоугольниками, перекрывающими указанный прямоугольник:

```
tarantool> rtree_index:select({0, 0, 10, 2}, {iterator='overlaps'})
---
- - [1, [1, 1]]
  - [2, [2, 2, 3, 3]]
...
```

Итератор NEIGHBOR ищет все кортежи и упорядочивает их по расстоянию до заданной точки:

```
tarantool> for i=1,10 do
         >    for j=1,10 do
         >        my_space:insert{i*10+j, {i, j, i+1, j+1}}
         >    end
         > end
---
...

tarantool> rtree_index:select({1, 1}, {iterator = 'neighbor', limit = 5})
---
- - [11, [1, 1, 2, 2]]
  - [12, [1, 2, 2, 3]]
  - [21, [2, 1, 3, 2]]
  - [22, [2, 2, 3, 3]]
  - [31, [3, 1, 4, 2]]
...
```

**Пример 2:**

3-мерные, 4-мерные или многомерные RTREE индексы используются так же, как и для двумерные, только пользователь должен указать больше координат в запросах. Вот небольшой пример использования для 4-мерного дерева:

```
tarantool> my_space = box.schema.create_space("tester")
tarantool> my_space:format{ { type = 'number', name = 'id' }, { type = 'array', name =
↪'content' } }
tarantool> primary_index = my_space:create_index('primary', { type = 'TREE', parts = {'id
↪'} })
tarantool> rtree_index = my_space:create_index('spatial', { type = 'RTREE', unique =␣
↪false, dimension = 4, parts = {'content'} })
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> my_space:insert{1, {1, 2, 3, 4}} -- вставка 4D-точки
tarantool> my_space:insert{2, {1, 1, 1, 1, 2, 2, 2, 2}} -- вставка 4D-тела

tarantool> rtree_index:select{1, 2, 3, 4} -- поиск определенной точки
---
- - [1, [1, 2, 3, 4]]
...

tarantool> rtree_index:select({0, 0, 0, 0, 3, 3, 3, 3}, {iterator = 'LE'}) -- выборка из␣
↪4D-тела
---
- - [2, [1, 1, 1, 1, 2, 2, 2, 2]]
...

tarantool> rtree_index:select({0, 0, 0, 0}, {iterator = 'neighbor'}) -- выборка соседей
---
- - [2, [1, 1, 1, 1, 2, 2, 2, 2]]
  - [1, [1, 2, 3, 4]]
...
```

**Примечание:** Не рекомендуется использовать с операцией select итератор NEIGHBOR без указанного значения `limit`. В этом случае будет получен спейс целиком, упорядоченный по возрастанию расстояния. Это может отразиться на производительности, поскольку в спейсе может храниться огромное количество данных.

Другая типичная ошибка — указать тип итератора без кавычек следующим образом: `rtree_index:select(rect, {iterator = LE})`. В этом случае `LE` представляет собой неопределенную переменную и обрабатывается как nil, поэтому итератор считается незаданным, и будет использован итератор по умолчанию EQ.

### BITSET-индексы

Bitset — это битовая маска. BITSET следует использовать для поиска по битовым маскам. Это может быть, например, сохранение вектора атрибутов и поиск по этим атрибутам.

**Предупреждение:** Currently, the *isolation level* of BITSET indexes in *MVCC transaction mode* is *read-committed* (not *serializable*, as stated). If a transaction uses these indexes, it can read committed or confirmed data (depending on the isolation level). However, the indexes are subject to different anomalies that can make them unserializable.

**Пример 1:**

Скрипт ниже показывает создание и поиск с помощью индекса BITSET. Обратите внимание, что BITSET не может быть уникальным, поэтому сначала создается индекс по первичному ключу, а битовые значения вводятся в шестнадцатеричном виде для удобства чтения.

```
tarantool> my_space = box.schema.space.create('space_with_bitset')
tarantool> my_space:create_index('primary_index', {
         >   parts = {1, 'string'},
```

(continues on next page)

(продолжение с предыдущей страницы)

```
         >   unique = true,
         >   type = 'TREE'
         > })
tarantool> my_space:create_index('bitset_index', {
         >   parts = {2, 'unsigned'},
         >   unique = false,
         >   type = 'BITSET'
         > })
tarantool> my_space:insert{'Tuple with bit value = 01', 0x01}
tarantool> my_space:insert{'Tuple with bit value = 10', 0x02}
tarantool> my_space:insert{'Tuple with bit value = 11', 0x03}
tarantool> my_space.index.bitset_index:select(0x02, {
         >   iterator = box.index.EQ
         > })
---
- - ['Tuple with bit value = 10', 2]
...
tarantool> my_space.index.bitset_index:select(0x02, {
         >   iterator = box.index.BITS_ANY_SET
         > })
---
- - ['Tuple with bit value = 10', 2]
  - ['Tuple with bit value = 11', 3]
...
tarantool> my_space.index.bitset_index:select(0x02, {
         >   iterator = box.index.BITS_ALL_SET
         > })
---
- - ['Tuple with bit value = 10', 2]
  - ['Tuple with bit value = 11', 3]
...
tarantool> my_space.index.bitset_index:select(0x02, {
         >   iterator = box.index.BITS_ALL_NOT_SET
         > })
---
- - ['Tuple with bit value = 01', 1]
...
```

**Пример 2:**

```
tarantool> box.schema.space.create('bitset_example')
tarantool> box.space.bitset_example:create_index('primary')
tarantool> box.space.bitset_example:create_index('bitset',{unique = false, type = 'BITSET
↪', parts = {2,'unsigned'}})
tarantool> box.space.bitset_example:insert{1,1}
tarantool> box.space.bitset_example:insert{2,4}
tarantool> box.space.bitset_example:insert{3,7}
tarantool> box.space.bitset_example:insert{4,3}
tarantool> box.space.bitset_example.index.bitset:select(2, {iterator = 'BITS_ANY_SET'})
```

Получим следующий результат:

```
---
- - [3, 7]
  - [4, 3]
...
```

поскольку (7 AND 2) не равно 0, и (3 AND 2) не равно 0.

Кроме того, есть *операции с итераторами с индексом*. Их можно использовать только с кодом на языках Lua и C/C++. Итераторы с индексом предназначены для обхода индексов по одному ключу за раз, поскольку используют особенности каждого типа индекса. Например, их можно использовать для оценки логических выражений при обходе BITSET-индексов или при обходе TREE-индексов в порядке по убыванию.

### Операции

#### Операции с данными

Tarantool поддерживает следующие основные операции с данными:

- пять операций по изменению данных (INSERT, UPDATE, UPSERT, DELETE, REPLACE) и
- одну операцию по выборке данных (SELECT).

Все они реализованы в виде функций во вложенном модуле *box.space*.

**Примеры:**

- *INSERT*: добавить новый кортеж к спейсу „tester“.

  Первое поле, field[1], будет 999 (тип MsgPack – `integer`, целое число).

  Второе поле, field[2], будет „Taranto“ (тип MsgPack – `string`, строка).

  ```
  tarantool> box.space.tester:insert{999, 'Taranto'}
  ```

- *UPDATE*: обновить кортеж, изменяя поле field[2].

  Оператор «{999}» со значением, которое используется для поиска поля, соответствующего ключу в первичном индексе, является обязательным, поскольку в запросе `update()` должен быть оператор, который указывает уникальный ключ, в данном случае – field[1].

  Оператор «{{„=“, 2, „Tarantino“}}» указывает, что назначение нового значения относится к field[2].

  ```
  tarantool> box.space.tester:update({999}, {{'=', 2, 'Tarantino'}})
  ```

- *UPSERT*: обновить или вставить кортеж, снова изменяя поле field[2].

  Синтаксис `upsert()` похож на синтаксис `update()`. Однако логика выполнения двух запросов отличается. UPSERT означает UPDATE или INSERT, в зависимости от состояния базы данных. Кроме того, выполнение UPSERT откладывается до коммита транзакции, поэтому в отличие от `update()`, `upsert()` не возвращает данные.

  ```
  tarantool> box.space.tester:upsert({999, 'Taranted'}, {{'=', 2, 'Tarantism'}})
  ```

- *REPLACE*: заменить кортеж, добавляя новое поле.

  Это действие также можно выполнить с помощью запроса `update()`, но обычно запрос `update()` более сложен.

```
tarantool> box.space.tester:replace{999, 'Tarantella', 'Tarantula'}
```

- *SELECT*: провести выборку кортежа.

  Оператор «{999}» все еще обязателен, хотя в нем не должен упоминаться первичный ключ.

```
tarantool> box.space.tester:select{999}
```

- *DELETE*: удалить кортеж.

  В этом примере мы определяем поле, соответствующее ключу в первичном индексе.

```
tarantool> box.space.tester:delete{999}
```

Подводя итоги по примерам:

- Функции `insert` и `replace` принимают кортеж (где первичный ключ – это часть кортежа).
- Функция `upsert` принимает кортеж (где первичный ключ – это часть кортежа), а также операции по обновлению.
- Функция `delete` принимает полный ключ любого уникального индекса (первичный или вторичный).
- Функция `update` принимает полный ключ любого уникального индекса (первичный или вторичный), а также операции к выполнению.
- Функция `select` принимает любой ключ: первичный/вторичный, уникальный/неуникальный, полный/часть.

Для получения более *подробной информации по использованию операций с данными* см. справочник по `box.space`.

---

**Примечание:** Помимо Lua можно использовать *коннекторы к Perl, PHP, Python или другому языку программирования*. Клиент-серверный протокол открыт и задокументирован. См. *БНФ с комментариями*.

---

### Факторы сложности

Во вложенных модулях *box.space* и *Вложенный модуль box.index* содержится информация о том, как факторы сложности могут повлиять на использование каждой функции.

| Фактор сложности | Эффект |
|---|---|
| Размер индекса | Количество ключей в индексе равно количеству кортежей в наборе данных. Для TREE-индекса: чем больше ключей, тем больше время поиска, хотя зависимость здесь, конечно же, нелинейная. Для HASH-индекса: чем больше ключей, тем больше нужно оперативной памяти, но количество низкоуровневых шагов остается примерно тем же. |
| Index type | Как правило, поиск по HASH-индексу работает быстрее, чем по TREE-индексу, если в спейсе более одного кортежа. |
| Количество обращений к индексам | Ordinarily, only one index is accessed to retrieve one tuple. But to update the tuple, there must be N accesses if the space has N different indexes.<br>Note regarding storage engine: Vinyl optimizes away such accesses if secondary index fields are unchanged by the update. So, this complexity factor applies only to memtx, since it always makes a full-tuple copy on every update. |
| Количество обращений к кортежам | Некоторые запросы, например SELECT, могут возвращать несколько кортежей. Обычно этот фактор менее важен, чем другие. |
| Настройки WAL | The important setting for the write-ahead log is *wal.mode*. If the setting causes no writing or delayed writing, this factor is unimportant. If the setting causes every data-change request to wait for writing to finish on a slow device, this factor is more important than all the others. |

### Описание схемы данных

В Tarantool использование схемы данных опционально.

При создании *спейса* схему можно не задавать и тогда в кортежах могут лежать произвольные данные. Это правило не распространяется на поля, по которым построены индексы. У таких полей данные должны быть одного типа.

Схему можно задать при создании спейса. Читайте подробнее в описании функции *box.schema.space.create()*. Если вы создали спейс без схемы, ее можно добавить позже с помощью метода *space_object:format()*.

После указания схемы данные начинают валидироваться по типам. Перед каждой операцией вставки или обновления они проверяются, и в случае несоответствия типов вы получите ошибку.

Мы рекомендуем использовать подход со схемой, потому что он помогает избежать ошибок.

Схему в Tarantool можно задавать двумя разными способами.

### Описание схемы в файле с кодом

Обычно файл с кодом называется `init.lua` и имеет следующее описание схемы:

```
box.cfg()

users = box.schema.create_space('users', { if_not_exists = true })
users:format({{ name = 'user_id', type = 'number'}, { name = 'fullname', type = 'string'}
↪})

users:create_index('pk', { parts = { { field = 'user_id', type = 'number'}}})
```

Этот подход довольно простой. Когда вы запустите tarantool, этот код исполнится и создаст схему. Чтобы запустить файл, используйте следующую команду:

```
tarantool init.lua
```

Но это может показаться слишком сложным, если вы не собираетесь глубоко разбираться с языком Lua и его синтаксисом.

Пример возможной сложности: в фрагменте выше есть вызов функций с двоеточием: `users:format`. Он используется, чтобы передать переменную `users` в качестве первого аргумента функции `format`. Это аналог `self` в объектно-ориентированных языках.

Поэтому вам может быть удобно описать схему через YAML.

### Описание через модуль DDL

Модуль DDL позволяет декларативно описывать схему данных в YAML формате.

Схема будет выглядеть примерно вот так:

```
spaces:
  users:
    engine: memtx
    is_local: false
    temporary: false
    format:
    - {name: user_id, type: uuid, is_nullable: false}
    - {name: fullname, type: string,  is_nullable: false}
    - {name: bucket_id, type: unsigned, is_nullable: false}
    indexes:
    - name: user_id
      unique: true
      parts: [{path: user_id, type: uuid, is_nullable: false}]
      type: HASH
    - name: bucket_id
      unique: false
      parts: [{path: bucket_id, type: unsigned, is_nullable: false}]
      type: TREE
    sharding_key: [user_id]
    sharding_func: test_module.sharding_func
```

Этот вариант проще для старта: его проще использовать и не нужно вникать в язык Lua.

To use the DDL module, put the following Lua code into the file that you use to run Tarantool. This file is usually called `init.lua`.

```
local yaml = require('yaml')
local ddl = require('ddl')

box.cfg{}

local fh = io.open('ddl.yml', 'r')
local schema = yaml.decode(fh:read('*all'))
fh:close()
local ok, err = ddl.check_schema(schema)
if not ok then
    print(err)
end
local ok, err = ddl.set_schema(schema)
if not ok then
    print(err)
end
```

**Предупреждение:** Менять схему в самом DDL после ее применения нельзя. Для миграций есть несколько подходов — они описаны в разделе *Миграции*.

### Персистентность

To ensure data persistence, Tarantool records updates to the database in the so-called *write-ahead log (WAL)* files. When a power outage occurs or the Tarantool instance is killed incidentally, the in-memory database is lost. In such case, Tarantool restores the data from WAL files by reading them and redoing the requests. This is called the «recovery process». You can change the timing of the WAL writer or turn it off by setting the *wal.mode*.

Tarantool также сохраняет набор файлов со статическими снимками данных (*snapshots*). Файл со снимком — это дисковая копия всех данных в базе на какой-то момент. Вместо того, чтобы перечитывать все WAL-файлы с момента создания базы, Tarantool в процессе восстановления может загрузить самый свежий снимок и затем прочитать только те WAL-файлы, которые были созданы начиная с момента сохранения снимка. После создания новых снимков более ранние WAL-файлы могут быть удалены, чтобы освободить место на диске.

To force immediate creation of a snapshot file, use the *box.snapshot()* function. To enable the automatic creation of snapshot files, use Tarantool's *checkpoint daemon*. The checkpoint daemon sets intervals for forced snapshots. It makes sure that the states of both memtx and vinyl storage engines are synchronized and saved to disk, and automatically removes earlier WAL files.

Файлы со снимками можно создавать, даже если WAL-файлы отсутствуют.

---

**Примечание:** The memtx engine takes only regular snapshots with the interval set in the checkpoint daemon configuration.

Движок vinyl постоянно сохраняет состояние в контрольной точке в фоновом режиме.

---

See the *Internals* section for more details about the WAL writer and the recovery process. To learn more about the configuration of the checkpoint daemon and WAL, check the *Persistence* page.

### Миграции

**Миграцией** называется любое изменение схемы данных: добавление/удаление поля, добавление/удаление индекса, изменение формата поля и т. д.

В Tarantool есть только два вида миграции схемы, при которых не нужно мигрировать данные:

- добавление поля в конец спейса
- добавление индекса

---

**Примечание:** Check the *Upgrading space schema* section. With the help of `space:upgrade()`, you can enable compression and migrate, including already created tuples.

---

#### Добавление поля в конец спейса

Добавление поля происходит следующим образом:

```
local users = box.space.users
local fmt = users:format()

table.insert(fmt, { name = 'age', type = 'number', is_nullable = true })
users:format(fmt)
```

Обратите внимание: поле обязательно должно иметь параметр `is_nullable`. Иначе произойдет ошибка.

После создания нового поля, вы, вероятно, захотите заполнить его данными. Для этой задачи удобно использовать модуль tarantool/moonwalker. В README есть описание того, как начать его использовать.

#### Добавление индекса

Про добавление индекса рассказывается в описании метода *space_object:create_index()*.

#### Остальные виды миграций

Остальные виды миграций также возможны, однако поддерживать консистентность данных при этом сложнее.

Миграции можно выполнять в двух случаях:

- при старте Tarantool, пока ни один клиент еще не использует БД
- в процессе обработки запросов, когда у БД уже есть активные клиенты

Для первого случая достаточно кода миграции, который будет написан и оттестирован. Самая сложная задача — это провести миграцию данных при наличии активных клиентов. Это необходимо иметь в виду при начальном проектировании схемы данных.

Мы выделяем следующие проблемы при наличии активных клиентов:

- Связанные данные могут атомарно изменяться.
- Нужно уметь отдавать данные как по новой схеме, так и по старой.

- Когда данные перемещают в новый спейс, доступ к данным должен учитывать, что данные могут быть как в одном спейсе, так и в другом.
- Запросы на запись не должны мешать корректной миграции. Типичный подход: всегда писать по новой схеме.

Эти проблемы могут быть или не быть релевантными в зависимости от вашего приложения и требований к его доступности.

#### Что нужно знать при написании сложных миграций

В Tarantool есть механизм транзакций. Он полезен при написании миграции, так как позволяет атомарно работать с данными. Но перед использованием механизма транзакций нужно знать об ограничениях.

Подробнее о них описано в разделе про *транзакции*.

#### Как применять миграции

Код миграций исполняется на запущенном инстансе Tarantool. Важно: ни один способ не гарантирует вам транзакционное применение миграций на всем кластере.

**Способ 1**: написать миграции в коде основного приложения

Это довольно просто: когда вы перезагружаете код, в нужный момент происходит миграция данных и схема базы данных обновляется. Однако такой способ может подойти не всем. У вас может не быть возможности перезапустить Tarantool или обновить код через механизм горячей перезагрузки (hot reload).

**Способ 2**: tarantool/migrations (только для кластера на Tarantool Cartridge)

Про этот способ подробнее написано в README самого модуля tarantool/migrations.

---

**Примечание:** Есть также два способа, которые мы **не рекомендуем использовать**, но вы можете найти их полезными по тем или иным причинам.

**Method 3**: the *tt* utility

Connect to the necessary instance using `tt connect`.

```
$ tt connect admin:password@localhost:3301
```

- Если ваша миграция написана в Lua файле, вы можете исполнить его с помощью функции `dofile()`. Вызовите ее и первым аргументом укажите путь до файла с миграцией. Выглядит это вот так:

  ```
  tarantool> dofile('0001-delete-space.lua')
  ---
  ...
  ```

- (или) можно скопировать код скрипта миграции, вставить его в консоль и применить.

You can also connect to the instance and execute the migration script in a single call:

```
$ tt connect admin:password@localhost:3301 -f 0001-delete-space.lua
```

**Способ 4**: применение миграции с помощью Ansible

Если вы используете Ansible-роль для развёртывания кластера Tarantool, то вы можете применить `eval`. Прочитать подробнее про `eval` можно в документации по Ansible-роли.

### 4.8.3 Fibers, yields, and cooperative multitasking

Creating a fiber is the Tarantool way of making application logic work in the background at all times. A **fiber** is a set of instructions that are executed with *cooperative multitasking*: the instructions contain *yield* signals, upon which control is passed to another fiber.

#### Fibers

Fibers are similar to threads of execution in computing. The key difference is that threads use preemptive multitasking, while fibers use cooperative multitasking (see *below*). This gives fibers the following two advantages over threads:

- Better controllability. Threads often depend on the kernel's thread scheduler to preempt a busy thread and resume another thread, so preemption may occur unpredictably. Fibers *yield* themselves to run another fiber while executing, so yields are controlled by application logic.
- Higher performance. Threads require more resources to preempt as they need to address the system kernel. Fibers are lighter and faster as they don't need to address the kernel to yield.

Yet fibers have some limitations as compared with threads, the main limitation being no multi-core mode. All fibers in an application belong to a single thread, so they all use the same CPU core as the parent thread. Meanwhile, this limitation is not really serious for Tarantool applications, because a typical bottleneck for Tarantool is the HDD, not the CPU.

A fiber has all the features of a Lua coroutine and all programming concepts that apply for Lua coroutines will apply for fibers as well. However, Tarantool has made some enhancements for fibers and has used fibers internally. So, although the use of coroutines is possible and supported, the use of fibers is recommended.

Any live fiber can be in one of three states: `running`, `suspended`, and `ready`. After a fiber dies, the `dead` status returns.

To learn more about fibers, go to the *fiber* module documentation.

#### Yields

Yield is an action that occurs in a *cooperative* environment that transfers control of the thread from the current fiber to another fiber that is ready to execute.

Any live fiber can be in one of three states: `running`, `suspended`, and `ready`. After a fiber dies, the `dead` status is returned. By observing fibers from the outside, you can only see `running` (for the current fiber) and `suspended` for any other fiber waiting for an event from the event loop (`ev`) for execution.

After a yield has occurred, the next `ready` fiber is taken from the queue and executed. When there are no more `ready` fibers, execution is transferred to the event loop.

After a fiber has yielded and regained control, it immediately issues *testcancel*.

Yields can be *explicit* or *implicit*.

### Explicit yields

**Explicit yields** are clearly visible from the invoking code. There are only two explicit yields: *fiber.yield()* and *fiber.sleep(t)*.

- *fiber.yield()* yields execution to another `ready` fiber while putting itself in the `ready` state, meaning that it will be executed again as soon as possible while being polite to other fibers waiting for execution.
- *fiber.sleep(t)* yields execution to another `ready` fiber and puts itself in the `suspended` state for time `t` until time passes and the event loop wakes up this fiber to the `ready` state.

In general, it is good behavior for long-running cpu-intensive tasks to yield periodically to be *cooperative* to other waiting fibers.

### Implicit yields

On the other hand, there are many operations, such as operations with sockets, file system, and disk I/O, which imply some waiting for the current fiber while others can be executed. When such an operation occurs, a possible blocking operation would be passed into the event loop and the fiber would be suspended until the resource is ready to continue fiber execution.

Here is the list of implicitly yielding operations:

- Connection establishment (*socket*).
- Socket read and write (*socket*).
- Filesystem operations (from *fio*).
- Channel data transfer (*fiber.channel*).
- File input/output (from *fio*).
- Console operations (since console is a socket).
- HTTP requests (since HTTP is a socket operation).
- Database modifications (if they imply a disk write).
- Database reading for the *vinyl* engine.
- Invocation of another process (*popen*).

---

**Примечание:** Please note that all operations of the `os` module are non-cooperative and exclusively block the whole tx thread.

---

For *memtx*, since all data is in memory, there is no yielding for a read request (like `:select`, `:pairs`, `:get`).

For *vinyl*, since some data may not be in memory, there may be disk I/O for a read (to fetch data from disk) or write (because a stall may occur while waiting for memory to be freed).

For both *memtx* and *vinyl*, since data change requests must be recorded in the *WAL*, there is normally a *box.commit()*.

With the default `autocommit` mode the following operations are yielding:

- *space:alter*.
- *space:drop*.
- *space:create_index*.

- *space:truncate*.
- *space:insert*.
- *space:replace*.
- *space:update*.
- *space:upserts*.
- *space:delete*.
- *index:update*.
- *index:delete*.
- *index:alter*.
- *index:drop*.
- *index:rename*.
- *box.commit* (*if there were some modifications within the transaction*).

To provide atomicity for transactions in transaction mode, some changes are applied to the modification operations for the *memtx* engine. After executing *box.begin* or within a *box.atomic* call, any modification operation will not yield, and yield will occur only on *box.commit* or upon return from *box.atomic*. Meanwhile, *box.rollback* does not yield.

That is why executing separate commands like `select()`, `insert()`, `update()` in the console inside a transaction without MVCC will cause it to an abort. This is due to implicit yield after each chunk of code is executed in the console.

### Example #1

- Engine = memtx.

```
space:get()
space:insert()
```

The sequence has one yield, at the end of the insert, caused by implicit commit; `get()` has nothing to write to the *WAL* and so does not yield.

- Engine = memtx.

```
box.begin()
space1:get()
space1:insert()
space2:get()
space2:insert()
box.commit()
```

The sequence has one yield, at the end of the `box.commit`, none of the inserts are yielding.

- Engine = vinyl.

```
space:get()
space:insert()
```

The sequence has one to three yields, since `get()` may yield if the data is not in the cache, `insert()` may yield if it waits for available memory, and there is an implicit yield at commit.

- Engine = vinyl.

```
box.begin()
space1:get()
space1:insert()
space2:get()
space2:insert()
box.commit()
```

The sequence may yield from 1 to 5 times.

### Example #2

Assume that there are tuples in the *memtx* space `tester` where the third field represents a positive dollar amount.

Let's start a transaction, withdraw from tuple#1, deposit in tuple#2, and end the transaction, making its effects permanent.

```
tarantool> function txn_example(from, to, amount_of_money)
         >   box.atomic(function()
         >     box.space.tester:update(from, {{'-', 3, amount_of_money}})
         >     box.space.tester:update(to,   {{'+', 3, amount_of_money}})
         >   end)
         >   return "ok"
         > end

Result:
---
...
tarantool> txn_example({999}, {1000}, 1.00)
---
- "ok"
...
```

If *wal_mode* = `none`, then there is no implicit yielding at the commit time because there are no writes to the *WAL*.

If a request if performed via network connector such as *net.box* and implies sending requests to the server and receiving responses, then it involves network I/O and thus implicit yielding. Even if the request that is sent to the server has no implicit yield. Therefore, the following sequence causes yields three times sequentially when sending requests to the network and awaiting the results.

```
conn.space.test:get{1}
conn.space.test:get{2}
conn.space.test:get{3}
```

#### Cooperative multitasking

Cooperative multitasking means that unless a running fiber deliberately *yields* control, it is not preempted by some other fiber. But a running fiber will deliberately yield when it encounters a «yield point»: a transaction commit, an operating system call, or an explicit «yield» request. Any system call which can block will be performed asynchronously, and any running fiber which must wait for a system call will be preempted, so that another ready-to-run fiber takes its place and becomes the new running fiber.

This model makes all programmatic locks unnecessary: cooperative multitasking ensures that there will be no concurrency around a resource, no race conditions, and no memory consistency issues. The way to achieve this is simple: Use no yields, explicit or implicit in critical sections, and no one can interfere with code execution.

For small requests, such as simple UPDATE or INSERT or DELETE or SELECT, fiber scheduling is fair: it takes little time to process the request, schedule a disk write, and yield to a fiber serving the next client.

However, a function may perform complex calculations or be written in such a way that yields take a long time to occur. This can lead to unfair scheduling when a single client throttles the rest of the system, or to apparent stalls in processing requests. It is the responsibility of the function author to avoid this situation. As a protective mechanism, a *fiber slice* can be used.

### 4.8.4 Транзакции

Transactions allow users to perform multiple operations atomically.

For more information on how transactions work in Tarantool, see the following sections:

#### Transaction model

#### Overview

The transaction model of Tarantool corresponds to the properties ACID (atomicity, consistency, isolation, durability).

Tarantool has two modes of transaction behavior:

- *Default* – suitable for fast monopolistic atomic transactions
- *MVCC* – designed for long-running concurrent transactions

Each transaction in Tarantool is executed in a single fiber on a single thread, sees a consistent database state and commits all changes atomically.

All transaction changes are written to the WAL (*Write Ahead Log*) in a single batch in a specific order at the time of the *commit*. If needed, transaction changes can also be rolled back – *completely* or to a specified *savepoint*.

Therefore, every transaction in Tarantool has the highest transaction isolation level – *serializable*.

### Isolation level

By *default*, the isolation level of Tarantool is *serializable*. The exception is a failure during writing to the WAL, which can occur, for example, when the disk space is over. In this case, the isolation level of the concurrent read transaction would be `read-committed`.

The *MVCC mode* provides several options that enable you to tune the visibility behavior during transaction execution.

### Read-committed

The `read-committed` isolation level makes visible all transactions that started commit (*box.commit()* was called).

- *Write transactions with reads*

  Manual usage of `read-committed` for write transactions with reads is completely safe, as this transaction will eventually result in a commit. If a previous transactions fails, this transaction will inevitably fail as well due to the *serializable* isolation level.

- *Read transactions*

  Manual usage of `read-committed` for read transactions may be unsafe, as it may lead to phantom reads.

### Read-confirmed

The `read-confirmed` isolation level makes visible all transactions that finished the commit (*box.commit()* was returned). This means that new data is already on disk or even on other replicas.

- *Read transactions*

  The use of `read-confirmed` is safe for read transactions given that data is on disk (for asynchronous replication) or even in other replicas (for synchronous replication).

- *Write transactions*

  To achieve *serializable*, any write transaction should read all data that has already been committed. Otherwise, it may conflict when it reaches its commit.

### Linearizable read

Linearizability of read operations implies that if a response for a write request arrived earlier than a read request was made, this read request should return the results of the write request. When called with `linearizable`, *box.begin()* yields until the instance receives enough data from remote peers to be sure that the transaction is linearizable.

Linearizable transactions may only perform requests to the following memtx space types:

- *synchronous*
- local (*created* with `is_local = true`)
- temporary (*created* with `temporary = true`)

A linearizable transaction can fail with an error in the following cases:

- If the node can't contact enough remote peers to determine which data is committed.

- If the data isn't received during the `timeout` specified in `box.begin()`.

---

**Примечание:** To start a linearizable transaction, the node should be the replication source for at least `N - Q + 1` remote replicas. Here `N` is the count of registered nodes in the cluster and `Q` is *replication_synchro_quorum*. So, for example, you can't perform a linearizable transaction on anonymous replicas because they can't be the source of replication for other nodes.

---

#### Best-effort (default)

To minimize the possibility of conflicts, MVCC uses what is called `best-effort` visibility:

- for write transactions, MVCC chooses *read-committed*
- for read transactions, MVCC chooses *read-confirmed*

This inevitably leads to the *serializable* isolation level. Since there is no option for MVCC to analyze the whole transaction to make a decision, it makes the choice on the first operation.

---

**Примечание:** If the *serializable* isolation level becomes unreachable, the transaction is marked as «conflicted» and rolled back.

---

### Thread model

#### Main threads

The thread model assumes that a *query* received by Tarantool via network is processed with three operating system **threads**:

1. The **network thread** (or *threads*) on the server side receives the query, parses the statement, checks if it is correct, and then transforms it into a special structure – a message containing an executable statement and its options.
2. The network thread sends this message to the instance's **transaction processor thread** (*TX thread*) via a lock-free message bus. Lua programs are executed directly in the transaction processor thread, and do not need to be parsed and prepared.

   The TX thread either uses a space index to find and update the tuple, or executes a stored function that performs a data operation.
3. The execution of the operation results in a message to the *write-ahead logging (WAL)* thread used to commit the transaction and the fiber executing the transaction is suspended. When the transaction results in a COMMIT or ROLLBACK, the following actions are taken:
   - The WAL thread responds with a message to the TX thread.
   - The fiber executing the transaction is resumed to process the result of the transaction.
   - The result of the fiber execution is passed to the network thread, and the network thread returns the result to the client.

---

**Примечание:** There is only one TX thread in Tarantool. Some users are used to the idea that there can be multiple threads working on the database. For example, thread #1 reads a row #x while thread #2 writes

a row #y. With Tarantool this does not happen. Only the TX thread can access the database, and there is only one TX thread for each Tarantool instance.

---

The TX thread can handle many *fibers* – a set of computer instructions that can contain «**yield**» signals. The TX thread executes all computer instructions up to a yield signal, and then switches to execute the instructions of another fiber.

*Yields* must happen, otherwise the TX thread would be permanently stuck on the same fiber.

### Supplementary threads

There are also several supplementary threads that serve additional capabilities:

- For *replication*, Tarantool creates a separate thread for each connected replica. This thread reads a write-ahead log and sends it to the replica, following its position in the log. Separate threads are required because each replica can point to a different position in the log and can run at different speeds.
- There is a thread pool for ad hoc asynchronous tasks, such as a DNS resolver or *fsync*.
- There is a thread pool that can be used for parallel sorting (hence, to parallelize building *indexes*). To configure it, use the *memtx.sort_threads* configuration option. The option sets the number of threads used to sort keys of secondary indexes on loading a `memtx` database.

  ---

  **Примечание:** Since *3.0.0*, this option replaces the approach when OpenMP threads are used to parallelize sorting. For backward compatibility, the `OMP_NUM_THREADS` environment variable is taken into account to set the number of sorting threads.

  ---

### Transaction mode: default

By default, Tarantool does not allow *«yielding»* inside a *memtx* transaction and the *transaction manager* is disabled. This allows fast atomic transactions without conflicts, but brings some limitations:

- You cannot use *interactive transactions*.
- Any fiber yield leads to the abort of a transaction.
- All changes are made immediately, but in the event of a yield or error, the transaction is rolled back, including the return of the previous data.

To learn how to enable yielding inside a *memtx* transaction, see *Transaction mode: MVCC*.

To switch back to the default mode, disable the transaction manager:

```
box.cfg { memtx_use_mvcc_engine = false }
```

### Transaction mode: MVCC

Since version *2.6.1*, Tarantool has another transaction behavior mode that allows *«yielding»* inside a *memtx* transaction. This is controlled by the *transaction manager*.

This mode allows concurrent transactions but may cause conflicts. You can use this mode on the *memtx* storage engine. The *vinyl* storage engine also supports MVCC mode, but has a different implementation.

---

**Примечание:** Currently, you cannot use several different storage engines within one transaction.

---

### Transaction manager

The transaction manager is designed to isolate concurrent transactions and provides a *serializable* transaction isolation level. It consists of two parts:

- *MVCC* – multi version concurrency control engine, which stores all change actions of all transactions. It also creates the transaction view of the database state and a read view (a fixed state of the database that is never changed by other transactions) when necessary.
- *Conflict manager* – a manager that tracks changes to transactions and determines their correctness in the serialization order. The conflict manager declares transactions to be in conflict or sends transactions to read views when necessary.

  Since version *2.10.1*, the conflict manager detects conflicts right after the first one of several conflicting transactions is committed. After this moment, any CRUD operations in the conflicted transaction will result in errors until the transaction is rolled back.

The transaction manager also provides a non-classical *snapshot* isolation level – this snapshot is not necessarily tied to the start time of the transaction, like the classical snapshot where a transaction can get a consistent snapshot of the database. The conflict manager decides if and when each transaction gets which snapshot. This avoids some conflicts compared to the classic snapshot isolation approach.

**Предупреждение:** Currently, the *isolation level* of BITSET and RTREE indexes in MVCC transaction mode is *read-committed* (not *serializable*, as stated). If a transaction uses these indexes, it can read committed or confirmed data (depending on the isolation level). However, the indexes are subject to different anomalies that can make them unserializable.

### Enabling the transaction manager

By default, the transaction manager is disabled. Use the *memtx_use_mvcc_engine* option to enable it via `box.cfg`.

```
box.cfg{memtx_use_mvcc_engine = true}
```

### Setting the transaction isolation level

The transaction manager has the following options for the *transaction isolation level*:

- `best-effort` (default)
- `read-committed`
- `read-confirmed`
- `linearizable` (only for a specific transaction)

Using `best-effort` as the default option allows MVCC to consider the actions of transactions independently and determine the best *isolation level* for them. It increases the probability of successful completion of the transaction and helps to avoid possible conflicts.

To set another default isolation level, for example, `read-committed`, use the following command:

```
box.cfg { txn_isolation = 'read-committed' }
```

Note that the `linearizable` isolation level can't be set as default and can be used for a specific transaction only. You can set an isolation level for a specific transaction in its `box.begin()` call:

```
box.begin({ txn_isolation = 'best-effort' })
```

In this case, you can also use the `default` option. It sets the transaction's isolation level to the one set in `box.cfg`.

---

**Примечание:** For autocommit transactions (actions with a statement without explicit `box.begin/box.commit` calls) there is a rule:

- Read-only transactions (for example, `select`) are performed with `read-confirmed`.
- All other transactions (for example, `replace`) are performed with `read-committed`.

---

You can also set the isolation level in the net.box *stream:begin()* method and *IPROTO_BEGIN* binary protocol request.

Choosing the better option depends on whether you have conflicts or not. If you have many conflicts, you should set a different option or use the *default transaction mode*.

### Examples with MVCC enabled and disabled

Create a file `init.lua`, containing the following:

```
fiber = require 'fiber'

box.cfg{ listen = '127.0.0.1:3301', memtx_use_mvcc_engine = false }
box.schema.user.grant('guest', 'super', nil, nil, {if_not_exists = true})

tickets = box.schema.create_space('tickets', { if_not_exists = true })
tickets:format({
    { name = "id", type = "number" },
    { name = "place", type = "number" },
})
tickets:create_index('primary', {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    parts = { 'id' },
    if_not_exists = true
})
```

Connect to the instance using the *tt connect* command:

```
tt connect 127.0.0.1:3301
```

Then try to execute the transaction with yield inside:

```
box.atomic(function() tickets:replace{1, 429} fiber.yield() tickets:replace{2, 429} end)
```

You will receive an error message:

```
---
- error: Transaction has been aborted by a fiber yield
...
```

Also, if you leave a transaction open while returning from a request, you will get an error message:

```
127.0.0.1:3301> box.begin()
     Failed to execute command: Transaction is active at return from function
```

Change `memtx_use_mvcc_engine` to `true`, restart Tarantool, and try again:

```
127.0.0.1:3301> box.atomic(function() tickets:replace{1, 429} fiber.yield()␣
↪tickets:replace{2, 429} end)
---
...
```

Now check if this transaction was successful:

```
127.0.0.1:3301> box.space.tickets:select({}, {limit = 10})
---
- - [1, 429]
  - [2, 429]
...
```

### Streams and interactive transactions

Since v. 2.10.0, IPROTO implements streams and interactive transactions that can be used when *memtx_use_mvcc_engine* is enabled on the server.

### Stream

A *stream* supports multiplexing several transactions over one connection. Each stream has its own identifier, which is unique within the connection. All requests with the same non-zero stream ID belong to the same stream. All requests in a stream are executed strictly sequentially. This allows the implementation of *interactive transactions*. If the stream ID of a request is `0`, it does not belong to any stream and is processed in the old way.

In *net.box*, a stream is an object above the connection that has the same methods but allows sequential execution of requests. The ID is automatically generated on the client side. If a user writes their own connector and wants to use streams, they must transmit the `stream_id` over the *IPROTO protocol*.

Unlike a thread, which involves multitasking and execution within a program, a stream transfers data via the protocol between a client and a server.

### Interactive transaction

An *interactive transaction* is one that does not need to be sent in a single request. There are multiple ways to begin, commit, and roll back a transaction, and they can be mixed. You can use *stream:begin()*, *stream:commit()*, *stream:rollback()* or the appropriate stream methods – `call`, `eval`, or `execute` – using the SQL transaction syntax.

Let's create a Lua client (`client.lua`) and run it with Tarantool:

```
local net_box = require 'net.box'
local conn = net_box.connect('127.0.0.1:3301')
local conn_tickets = conn.space.tickets
local yaml = require 'yaml'

local stream = conn:new_stream()
local stream_tickets = stream.space.tickets

-- Begin transaction over an iproto stream:
stream:begin()
print("Replaced in a stream\n".. yaml.encode(  stream_tickets:replace({1, 768}) ))

-- Empty select, the transaction was not committed.
-- You can't see it from the requests that do not belong to the
-- transaction.
print("Selected from outside of transaction\n".. yaml.encode(conn_tickets:select({},
↪{limit = 10}) ))

-- Select returns the previously inserted tuple
-- because this select belongs to the transaction:
print("Selected from within transaction\n".. yaml.encode(stream_tickets:select({},
↪{limit = 10}) ))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Commit transaction:
stream:commit()

-- Now this select also returns the tuple because the transaction has been committed:
print("Selected again from outside of transaction\n".. yaml.encode(conn_tickets:select({}
↪, {limit = 10}) ))

os.exit()
```

Then call it and see the following output:

```
Replaced in a stream
--- [1, 768]
...

Selected from outside of transaction
---
- [1, 429]
- [2, 429]
...

Selected from within transaction
---
- [1, 768]
- [2, 429]
...

Selected again from outside of transaction
---
- [1, 768]
- [2, 429]
...```
```

### 4.8.5 Modules

Any logic that is used in Tarantool can be packaged as an application or a reusable **module**. A module is an optional library that extends Tarantool functionality. It can be used by Tarantool applications or other modules. Modules allow for easier code management and hot code reload without restarting the Tarantool instance. Like applications, modules in Tarantool can be written in Lua, C, or C++. Lua modules are also referred to as «rocks».

For example, here is a Lua module named `mymodule.lua` that exports one function named `myfun`:

```
local exports = {}
exports.myfun = function(input_string)
   print('Hello', input_string)
end
return exports
```

To launch the function `myfun()` – from another module, from a Lua application, or from Tarantool itself, – save this module as a file, then load this module with the `require()` directive and call the exported function.

For example, here's a Lua application that uses `myfun()` from `mymodule.lua`:

```
-- loading the module
local mymodule = require('mymodule')

-- calling myfun() from within test()
local test = function()
  mymodule.myfun()
end
```

Tarantool provides an extensive library of compatible modules. Install them using Tarantool's CLI utility *tt*. Some modules are also included in the Tarantool repository and can be installed via your operating system's package manager.

Learn how to:

- *install a module*
- *contribute a module*

### 4.8.6 Sharding

С ростом проекта масштабируемость баз данных часто становится одной из наиболее серьезных проблем. Если отдельный сервер не может справиться с нагрузкой, необходимо применять средства масштабирования.

**Шардирование** – это архитектурый принцип баз данных, который позволяет их горизонтально масштабировать, что означает разбиение набора даных на части и распределение их по нескольким серверам.

С помощью модуля vshard кортежи набора данных распределяются по множеству узлов, на каждом из которых находится экземпляр сервера базы данных Tarantool. Каждый экземпляр обрабатывает лишь подмножество от общего количества данных, поэтому увеличение нагрузки можно компенсировать добавлением новых серверов. Первоначальный набор данных секционируется на множество частей, то есть каждая часть хранится на отдельном сервере.

Модуль `vshard` основан на концепции *виртуальных сегментов*: набор кортежей распределяется на большое количество абстрактных виртуальных узлов (**виртуальных сегментов**, или просто **сегментов** далее по тексту), а не на малое количество физических узлов.

Секционирование набора данных осуществляется с помощью **сегментных ключей** (идентификаторов сегментов). Хеширование сегментного ключа в большое количество сегментов позволяет незаметно для пользователя изменять количество серверов в кластере. **Механизм балансирования** распределяет сегменты между шардами при добавлении или удалении каких-либо серверов.

Для сегментов предусмотрены **состояния**, поэтому можно легко отслеживать состояние сервера. Например, активен ли экземпляр сервера и доступен ли он для всех типов запросов, или же произошел отказ, и сервер принимает только запросы на чтение.

Модуль `vshard` предоставляет общедоступные и внутренние API роутера и хранилища для приложений с поддержкой шардинга.

Начните с *руководства по быстрому запуску* или узнайте подробности об *архитектуре шардирования* в Tarantool:

### Архитектура

#### Общие сведения

Рассмотрим распределенный Tarantool-кластер, состоящий из подкластеров под названием **шарды**, в каждом из которых хранится некоторая часть данных. Каждый шард, в свою очередь, представляет собой **набор реплик**, одна из которых служит ведущим узлом, обрабатывающим все запросы на чтение и запись.

Весь набор данных при шардинге распределяется на заданное количество *виртуальных сегментов* (далее по тексту просто **сегменты**). Каждому из них присваивается уникальный номер от 1 до N, где N – это общее количество сегментов. Специально выбирается количество сегментов на несколько порядков больше, чем потенциальное количество кластерных узлов даже с учетом будущего масштабирования кластера. Например, если предполагается M узлов, набор данных может быть разделен на 100 * M или даже 1000 * M сегментов. Особое внимание следует уделить выбору количества сегментов: слишком большое число может потребовать дополнительную память для хранения информации о маршрутизации; слишком маленькое может привести к снижению степени детализации балансировки.

Каждый шард хранит уникальное подмножество сегментов. Один сегмент не может относиться к нескольким шардам одновременно, как показано на схеме ниже:

Такая схема распределения сегментов по шардам хранится в таблице в одном из системных пространств Tarantool'а, при этом в каждом шарде содержится только определенную часть схемы, которая покрывает присвоенные этому шарду сегменты.

Помимо таблицы, **идентификатор сегмента** также хранится в специальном поле каждого кортежа каждой таблицы, участвующей в шардинге.

Как только шард получает любой запрос (за исключением SELECT) от приложения, этот шард сверяет идентификатор сегмента, указанный в запросе, с таблицей идентификаторов сегментов, которые принадлежат данному узлу. Если указанный идентификатор сегмента недействителен, то запрос завершается со следующей ошибкой: «wrong bucket" (неверный сегмент). В противном случае запрос выполняется, и всем создаваемым данным присваивается указанный в запросе идентификатор сегмента. Обратите внимание, что запрос должен изменять только данные с тем же идентификатором сегмента, что и в запросе.

Хранение идентификаторов сегментов как в самих данных, так и в таблице обеспечивает согласованность данных независимо от логики приложения и прозрачность балансировки для приложения. Хранение таблицы соответствий в системном спейсе обеспечивает последовательность шардинга в случае восстановления после отказа, так как у всех реплик в шарде будет одно исходное состояние таблицы.

#### Виртуальные сегменты

Набор данных при шардинге распределяется на большое количество абстрактных узлов, которые называются **виртуальные сегменты** (далее по тексту просто **сегменты**).

Секционирование набора данных происходит с помощью сегментного ключа (или **идентификатора сегмента** (bucket id) в терминах Tarantool). Идентификатор сегмента – это число от 1 до N, где N – это общее количество сегментов.

В каждом наборе реплик есть уникальное подмножество сегментов. Один сегмент не может относиться к нескольким наборам реплик одновременно.

Общее количество сегментов определяет администратор, который настраивает первоначальную конфигурацию кластера.

В каждом спейсе, который будет разделен на шарды, должно быть числовое поле с идентификаторами сегментов. Это поле должно соответствовать следующим требованиям:

- Тип данных поля может быть: unsigned (без знака), number (число) или integer (целое число).
- Поле не должно быть нулевым.
- Поле должно быть проиндексировано с помощью *shard_index*. Имя по умолчанию для этого индекса: `bucket_id`.

См. *пример конфигурации*.

### Структура

Сегментированный кластер в Tarantool состоит из:

- хранилищ,
- роутеров
- и балансировщика.

### Хранилище

**Хранилище** (storage) – это узел, который хранит подмножество набора данных. Несколько реплицируемых (для резерва) хранилищ составляют **набор реплик** (также называемый **шардом**).

У каждого хранилища в наборе реплик есть роль: **мастер** или **реплика**. Мастер обрабатывает запросы на чтение и запись. Реплика обрабатывает запросы на чтение, но не может обрабатывать запросы на запись.

### Роутер

**Роутер** (router) – это автономный компонент ПО, который обеспечивает маршрутизацию запросов чтения и записи от клиентского приложения к шардам.

Все запросы из приложения приходят в сегментированный кластер через роутер (`router`). Роутер сохраняет топологию сегментированного кластера прозрачной для приложения, не сообщая приложению:

- номер и местоположение шардов,
- процесс балансировки данных,
- наличие отказа и восстановление после отказа реплики.

Роутер также может самостоятельно вычислить идентификатор сегмента при условии, что приложение четко определяет правила вычисления идентификатора сегмента на основе данных запроса. Для этого роутеру необходимо знать схему данных.

У роутера нет постоянного статуса, он не хранит топологию кластера и не выполняет балансировку данных. Роутер – это автономный компонент ПО, который может работать на уровне хранилища или на уровне приложения в зависимости от функций приложения.

Роутер поддерживает постоянный пул соединений со всеми хранилищами, созданными при запуске, что помогает избежать ошибок конфигурации. После создания пула роутер кэширует текущее состояние таблицы `_vbucket`, чтобы ускорить маршрутизацию. Если сегмент был перемещен в другое хранилище в результате балансировки, или же один из шардов переключается на реплику, роутер обновит таблицу маршрутизации так, чтобы это было понятно приложению.

Шардинг не интегрирован ни в одну систему централизованного хранения конфигураций. Предполагается, что само приложение обрабатывает взаимодействие с такой системой и передает параметры шардинга. При этом конфигурацию можно изменить динамически, например, при добавлении или удалении одного или нескольких шардов:

1. Чтобы добавить новый шард в кластер, системный администратор сначала изменяет конфигурацию всех роутеров, а затем конфигурацию всех хранилищ.
2. Новый шард становится доступен для балансировки на уровне хранилища.
3. В результате балансировки один из виртуальных сегментов перемещается на новый шард.
4. При попытке доступа к виртуальному сегменту роутер получает специальный код ошибки, который указывает новое местоположение сегмента.

### CRUD-операции: create, read, update, delete (создание, чтение, изменение, удаление)

CRUD-операции могут:

- либо выполняться в рамках хранимой процедуры в хранилище,
- либо запускаться приложением.

В любом случае приложение должно включать идентификатор рабочего сегмента в запрос. При выполнении запроса вставки INSERT идентификатор сегмента хранится в созданном кортеже. В других случаях проверяется, совпадает ли указанный идентификатор рабочего сегмента с идентификатором сегмента кортежа, в который вносятся изменения.

### SELECT-запросы

Поскольку хранилище не знает о соответствии идентификатора сегмента и первичного ключа, все запросы выборки SELECT в хранимых процедурах внутри хранилища выполняются только локально. SELECT-запросы, которые были инициализированы приложением, направляются на роутер. И если приложение передало идентификатор сегмента, роутер использует его для вычисления шарда.

### Вызов хранимых процедур

Существует несколько способов вызвать хранимые процедуры в наборах реплик кластера. Хранимые процедуры можно вызвать:

- либо на определенном виртуальном сегменте, расположенном в наборе реплик (в этом случае необходимо различать процедуры чтения и записи, так как процедуры записи не применимы к перемещаемым сегментам),
- либо без указания определенного сегмента.

Все проверки правильности маршрутизации, выполняемые для шардированных DML-операций, распространяются и на хранимые процедуры, связанные с сегментами.

### Балансировщик

**Балансировщик** представляет собой фоновый процесс балансировки, который обеспечивает равномерное распределение сегментов по шардам. Во время балансировки происходит миграция сегментов по наборам реплик.

The rebalancer «wakes up» periodically and redistributes data from the most loaded nodes to less loaded nodes. Rebalancing starts if the **replicaset disbalance** of a replica set exceeds a disbalance threshold specified in the configuration.

The replicaset disbalance is calculated as follows:

```
|эталонное_число_сегментов - текущее_число_сегментов| / эталонное_число_сегментов * 100
```

### Миграция сегментов

Набор реплик, из которого переносится сегмент, называется **исходный** (source); а набор реплик, куда переносится сегмент, называется **целевой** (destination).

**Блокировка набора реплик** позволяет набору реплик оставаться невидимым для балансировщика. Набор реплик с блокировкой не может ни принимать новые сегменты, ни мигрировать свои собственные.

Во время миграции у сегмента могут быть разные статусы:

- ACTIVE (активный) – сегмент доступен для запросов чтения и записи.
- PINNED (закрепленный) – сегмент заблокирован для миграции в другой набор реплик. Во всем остальном закрепленные сегменты аналогичны активным сегментам.
- SENDING (отправляемый) – в настоящий момент сегмент копируется в целевой набор реплик; запросы на чтение в исходный набор реплик обрабатываются.
- RECEIVING (принимающий) – происходит наполнение сегмента; все запросы отклоняются.
- SENT (отправленный) – сегмент был перенесен в целевой набор реплик. Роутер использует статус SENT, чтобы определить новое местонахождение сегмента. Сегмент в статусе SENT переходит в статус мусора GARBAGE автоматически через количество секунд, указанное в BUCKET_SENT_GARBAGE_DELAY, по умолчанию равное *0,5 секунды*.
- GARBAGE (мусор) – произошла миграция сегмента в целевой набор реплик во время балансировки; или же принимающий сегмент был в статусе RECEIVING, но произошла ошибка во время миграции.

Сегменты в статусе мусора GARBAGE удаляются сборщиком мусора.

Миграция происходит следующим образом:

1. В целевом наборе реплик создается новый сегмент, который получает статус RECEIVING (принимающий), начинается копирование данных, и сегмент отклоняет все запросы.
2. Отправляемый сегмент в исходном наборе реплик получает статус SENDING и продолжает обрабатывать запросы на чтение.
3. После копирования данных сегмент в исходном наборе реплик получает статус отправленного (SENT) и перестает принимать запросы.
4. Сегмент в целевом наборе реплик переходит в активный статус (ACTIVE) и начинает принимать все запросы.

**Примечание:** Есть специальная ошибка `vshard.error.code.TRANSFER_IS_IN_PROGRESS`, которая возвращается в том случае, если запрос пытается выполнить действие, неприменимое к перемещаемому сегменту. В этом случае необходимо повторить попытку выполнения запроса.

### Системный спейс _bucket

Системный спейс `_bucket` в каждом наборе реплик хранит идентификаторы сегментов данного набора реплик. Спейс содержит следующие поля:

- `bucket` – идентификатор сегмента
- `status` – статус сегмента
- `destination` – UUID целевого набора реплик

Пример `_bucket.select{}`:

```
---
- - [1, ACTIVE, abfe2ef6-9d11-4756-b668-7f5bc5108e2a]
  - [2, SENT, 19f83dcb-9a01-45bc-a0cf-b0c5060ff82c]
...
```

После миграции сегмента UUID целевого набора реплик вносится в таблицу. Пока сегмент еще находится в исходном наборе реплик, значение UUID целевого набора реплик равно `NULL`.

### Таблица маршрутизации

Таблица маршрутизации роутера отображает все идентификаторы сегментов с соответствующими наборами реплик. Она обеспечивает консистентность шардинга в случае отказа.

Роутер поддерживает постоянный пул соединений со всеми хранилищами, созданными при запуске, что помогает избежать ошибки конфигурации. После создания пула соединений роутер кэширует текущее состояние таблицы маршрутизации, чтобы ускорить ее. Если произошла миграция сегмента в другое хранилище после балансировки или же отказ, который вызвал переключение шарда на другую реплику, файбер обнаружения (`discovery fiber`) в роутере обновит таблицу маршрутизации автоматически.

Поскольку идентификатор сегмента явно указан как в данных, так и в таблице отображения на роутере, данные сохраняются независимо от логики приложения. Это также обеспечивает прозрачность балансировки для приложения.

### Обработка запросов

Запросы в базу данных можно производить из приложения или с помощью хранимых процедур. В любом случае идентификатор сегмента следует явным образом указать в запросе.

Сначала все запросы направляются в роутер. Роутер поддерживает только операцию вызова, которая выполняется с помощью функции `vshard.router.call()`:

```
result = vshard.router.call(<идентификатор_сегмента>, <режим>, <имя_функции>, {<список_
↪аргументов>}, {<опции>})
```

Запросы обрабатываются следующим образом:

1. Роутер использует идентификатор сегмента для поиска набора реплик с соответствующим сегментом в таблице маршрутизации.

   Если роутер не содержит информацию о соответствии идентификатора сегмента набору реплик (файбер обнаружения еще не заполнил таблицу), роутер выполняет запросы ко всем хранилищам, чтобы обнаружить местонахождение сегмента.

2. После обнаружения сегмента шард проверяет:

   - хранится ли сегмент в системном спейсе `_bucket` набора реплик;
   - находится ли сегмент в статусе ACTIVE (активный) или PINNED (закрепленный) (если выполняется запрос на чтение, то сегмент может находиться в состоянии отправки SENDING).

3. Если проверка пройдена, запрос выполняется. В противном случае, выполнение запроса прекращается с ошибкой: **“`wrong bucket`”** (несоответствующий сегмент).

### Глоссарий

**Вертикальное масштабирование** Добавление мощности в отдельный сервер: использование более мощного процессора, добавление оперативной памяти, добавление хранилищ и т.д.

**Горизонтальное масштабирование** Добавление дополнительных серверов в пул ресурсов, последующее секционирование и распределение набора данных по серверам.

**Шардинг** Архитектура базы данных, которая допускает секционирование набора данных по сегментному ключу и распределение набора данных по нескольким серверам. Шардинг представляет собой частный случай горизонтального масштабирования.

**Узел** Виртуальный или физический экземпляр сервера.

**Кластер** Набор узлов, которые составляют отдельную группу.

**Хранилище** Узел, который хранит подмножество данных из набора.

**Набор реплик** Ряд узлов, на которых хранятся копии набора данных. У каждого хранилища в наборе реплик есть роль: мастер или реплика.

**Мастер** Хранилище в наборе реплик, которое обрабатывает запросы на чтение и запись.

**Реплика** Хранилище в наборе реплик, которое обрабатывает только запросы на чтение.

**Запросы на чтение** Запросы только на чтение, то есть выборка.

**Запросы на запись** Операции по изменению данных, то есть запросы на создание, чтение, изменение и удаление данных.

**Сегменты (виртуальные сегменты)** Абстрактные виртуальные узлы, на которые производится секционирование набора данных по сегментному ключу (идентификатору сегмента).

**Идентификатор сегмента** Сегментный ключ, который определяет принадлежность сегмента к определенному набору реплик. Идентификатор сегмента можно вычислить по *хеш-ключу*.

**Роутер** Прокси-сервер, который отвечает за запросы маршрутизации от приложения к узлам в кластере.

Ознакомьтесь с *руководством администратора* по шардированию или перейдите к *справочнику* по конфигурации и API модуля `vshard`.

### 4.8.7 Replication

Механизм репликации позволяет сразу многим экземплярам Tarantool работать с копиями одних и тех же баз данных. При этом все базы остаются в синхронизированном состоянии благодаря тому, что каждый экземпляр может сообщать другим экземплярам о совершенных им изменениях.

This section includes the following topics:

#### Архитектура механизма репликации

#### Механизм репликации

#### Overview

A pack of instances that operate on copies of the same databases makes up a **replica set**. Each instance in a replica set has a role: **master** or **replica**.

A replica gets all updates from the master by continuously fetching and applying its *write-ahead log (WAL)*. Each record in the WAL represents a single Tarantool data-change request such as *INSERT*, *UPDATE*, or *DELETE*, and is assigned a monotonically growing log sequence number (**LSN**). In essence, Tarantool replication is **row-based**: each data-change request is fully deterministic and operates on a single *tuple*. However, unlike a classical row-based log, which contains entire copies of the changed rows, Tarantool's WAL contains copies of the requests. For example, for UPDATE requests, Tarantool only stores the primary key of the row and the update operations to save space.

---

**Примечание:** *WAL extensions* available in Tarantool Enterprise Edition enable you to add auxiliary information to each write-ahead log record. This information might be helpful for implementing a CDC (Change Data Capture) utility that transforms a data replication stream.

---

The following are specifics of adding different types of information to the WAL:

- Invocations of **stored programs** are not written to the WAL. Instead, records of the actual **data-change requests, performed by the Lua code**, are written to the WAL. This ensures that the possible non-determinism of Lua does not cause replication to go out of sync.
- Data definition operations on **temporary spaces** (*created* with `temporary = true`), such as creating/dropping, adding indexes, and truncating, are written to the WAL, since information about temporary spaces is stored in non-temporary system spaces, such as *box.space._space*.
- Data change operations on temporary spaces are not written to the WAL and are not replicated.
- Data change operations on **replication-local** spaces (*created* with `is_local = true`) are written to the WAL but are not replicated.

To learn how to enable replication, check the *Bootstrapping a replica set* guide.

### Replication stages

To create a valid initial state, to which WAL changes can be applied, every instance of a replica set requires a start set of *checkpoint files*, such as `.snap` files for memtx and `.run` files for vinyl. A replica goes through the following stages:

1. **Bootstrap** (optional)

   When an entire replica set is bootstrapped for the first time, there is no master that could provide the initial checkpoint. In such a case, replicas connect to each other and elect a master. The master creates the starting set of checkpoint files and distributes them to all the other replicas. This is called an **automatic bootstrap** of a replica set.

2. **Join**

   At this stage, a replica downloads the initial state from the master. The master register this replica in the *box.space._cluster* space. If join fails with a non-critical *error*, for example, `ER_READONLY`, `ER_ACCESS_DENIED`, or a network-related issue, an instance tries to find a new master to join.

   ---

   **Примечание:** On subsequent connections, a replica downloads all changes happened after the latest local LSN (there can be many LSNs – each master has its own LSN).

   ---

3. **Follow**

   At this stage, a replica fetches and applies updates from the master's write-ahead log.

You can use the *box.info.replication[n].upstream.status* property to monitor the status of a replica.

### Replica set and instance UUIDs

Each replica set is identified by a globally unique identifier, called the **replica set UUID**. The identifier is created by the master, which creates the very first checkpoint and is part of the checkpoint file. It is stored in the *box.space._schema* system space, for example:

```
tarantool> box.space._schema:select{'cluster'}
---
- - ['cluster', '6308acb9-9788-42fa-8101-2e0cb9d3c9a0']
...
```

Additionally, each instance in a replica set is assigned its own UUID, when it joins the replica set. It is called an **instance UUID** and is a globally unique identifier. The instance UUID is checked to ensure that instances do not join a different replica set, e.g. because of a configuration error. A unique instance identifier is also necessary to apply rows originating from different masters only once, that is, to implement multi-master replication. This is why each row in the write-ahead log, in addition to its log sequence number, stores the instance identifier of the instance on which it was created. But using a UUID as such an identifier would take too much space in the write-ahead log, thus a shorter integer number is assigned to the instance when it joins a replica set. This number is then used to refer to the instance in the write-ahead log. It is called **instance ID**. All identifiers are stored in the system space *box.space._cluster*, for example:

```
tarantool> box.space._cluster:select{}
---
- - [1, '88580b5c-4474-43ab-bd2b-2409a9af80d2']
...
```

Здесь ID экземпляра – 1 (уникальный номер в рамках набора реплик), а UUID экземпляра – 88580b5c-4474-43ab-bd2b-2409a9af80d2 (глобально уникальный).

Использование идентификаторов экземпляра также полезно для отслеживания состояния всего набора реплик. Например, *box.info.vclock* описывает состояние репликации в отношении каждого подключенного узла.

```
tarantool> box.info.vclock
---
- {1: 827, 2: 584}
...
```

Here vclock contains log sequence numbers (827 and 584) for instances with instance IDs 1 and 2.

If required, you can explicitly specify the instance and the replica set UUID values rather than letting Tarantool generate them. To learn more, see the *replicaset_uuid* configuration parameter description.

### Роли в репликации: мастер и реплика

Конфигурационный параметр *read_only* определяет роль в репликации (мастер или реплика). Рекомендованная роль для всех экземпляров в наборе реплик, кроме одного – «read-only» (реплика).

В конфигурации мастер-реплика каждое изменение, сделанное на мастере, будет отображаться на репликах, но не наоборот.

Простой набор реплик с двумя экземплярами, один из которых является мастером и расположен на одной машине, а другой – реплика – расположен на другой машине, дает два преимущества:

- **failover**, because if the master goes down, then the replica can take over, and
- **балансировка нагрузки**, потому что клиенты во время запросов чтения могут подключаться к мастеру или к реплике.

В конфигурации **мастер-мастер** (которая также называется «многомастерной») каждое изменение на любом экземпляре будет также отображаться на другом.

Восстановление после отказа в таком случае также будет преимуществом, а балансировка нагрузки улучшится, поскольку любой экземпляр может обрабатывать запросы и на чтение, и на запись. В то же время, при многомастерной конфигурации необходимо понимать **гарантии репликации**, которые обеспечивает асинхронный протокол, внедренный в Tarantool.

Многомастерная репликация Tarantool гарантирует, что каждое изменение на каждом мастере передается на все экземпляры и применяется только один раз. Изменения с одного экземпляра применяются в том же порядке, что и на исходном экземпляре. Однако изменения с разных экземпляров могут смешиваться и применяться в различном порядке на разных экземплярах. В определенных случаях это может привести к рассинхронизации.

Например, принимая, что проводятся только операции добавления данных в базу (т.е. она содержит только вставки), многомастерная конфигурация сработает хорошо. Если данные также удаляются, но порядок операций удаления на разных репликах не играет важной роли (например, DELETE используется для отсечения устаревших данных), то конфигурация мастер-мастер также безопасна.

UPDATE operations, however, can easily go out of sync. For example, assignment and increment are not commutative and may yield different results if applied in a different order on different instances.

В общем смысле, безопасно использовать репликацию мастер-мастер в Tarantool, если все изменения в базе данных являются **коммутативными**: конечный результат не зависит от порядка, в котором применяются изменения. Дополнительную информацию о бесконфликтных типах реплицируемых данных можно получить здесь.

### Replication topologies: cascade, ring, and full mesh

Replication topology is set by the *replication* configuration parameter. The recommended topology is a **full mesh** because it makes potential failover easy.

Some database products offer **cascading replication** topologies: creating a replica on a replica. Tarantool does not recommend such a setup.

Недостаток каскадного набора реплик заключается в том, что некоторые экземпляры не подключаются к другим экземплярам, поэтому не могут получать от них изменения. Одно важное изменение, которое следует передавать на все экземпляры в наборе реплик – запись в системный спейс `box.space._cluster` с UUID набора реплик. Не зная UUID набора реплик, мастер отклоняет подключения от таких экземпляров при изменении топологии репликации. Вот как это может произойти:

We have a chain of three instances. Instance #1 contains entries for instances #1 and #2 in its `_cluster` space. Instances #2 and #3 contain entries for instances #1, #2, and #3 in their `_cluster` spaces.

Now instance #2 is faulty. Instance #3 tries connecting to instance #1 as its new master, but the master refuses the connection since it has no entry, for example, #3.

Тем не менее, **кольцевая топология** поддерживается:

Поэтому если необходима каскадная топология, можно первоначально создать кольцо, чтобы все экземпляры знали UUID друг друга, а затем разъединить цепочку в необходимом месте.

Как бы то ни было, для репликации мастер-мастер рекомендуется **полная ячеистая** топология:

В таком случае можно решить, где расположить экземпляры ячейки – в том же центре обработки данных или разместить в нескольких центрах. Tarantool будет автоматически следить за тем, что каждая строка применяется однократно на каждом экземпляре. Чтобы удалить экземпляр из ячейки после отказа, просто измените конфигурационный параметр `replication`.

Таким образом можно обеспечить доступность всего кластера в случае локального отказа, например отказа одного экземпляра в одном центре обработки данных, а также в случае отказа всего центра обработки данных.

Максимальное количество реплик в ячейке – 32.

### Статус orphan (одиночный)

During `box.cfg()`, an instance tries to join all nodes listed in *box.cfg.replication*. If the instance does not succeed in connecting to the required number of nodes (see *bootstrap_strategy*), it switches to the *orphan status*.

## Синхронная репликация

### Общие сведения

По умолчанию репликация в Tarantool **асинхронная**: локальный коммит транзакции на мастере не означает, что эта транзакция будет сразу же выполнена на репликах. Если мастер сообщит клиенту об успешном выполнении операции, а потом прекратит работу и после отказа восстановится на реплике, то с точки зрения клиента транзакция пропадет.

Эту проблему решает **синхронная** репликация. Каждая синхронная транзакция проходит коммит лишь после репликации на некотором количестве экземпляров, и только тогда клиенту приходит ответ о завершении транзакции.

To enable synchronous replication, use the *space_opts.is_sync* option when creating or altering a space.

### Синхронные и асинхронные транзакции

В Tarantool синхронную репликацию можно настроить для отдельных спейсов. Эта удобная функция позволит вам не потерять в производительности, если синхронная репликация нужна вам лишь изредка для изменения критически важных данных.

Если наряду с несколькими синхронными транзакциями, ожидающими репликации, совершается асинхронная транзакция, она блокируется синхронными. Коммиты при этом выполняются в той последовательности, в которой для каждой из транзакций вызывается метод *box.commit()*. Похожим образом работает обычная очередь асинхронных транзакций. Можно сформулировать **правило коммитов**: порядок коммитов соответствует порядку вызова `box.commit()` для каждой из транзакций, независимо от того, синхронные транзакции или асинхронные.

Если для одной из синхронных транзакций истечет время ожидания, эта транзакция будет отменена, а вместе с ней и все последующие транзакции в очереди на репликацию. Похожим образом отменяются и асинхронные транзакции при ошибке записи в WAL. Действует **правило отмены**: транзакции отменяются в порядке, обратном порядку вызова `box.commit()` для каждой из них, независимо от того, синхронные транзакции или асинхронные.

Асинхронная транзакция, заблокированная синхронной, не становится сама синхронной, а просто ожидает коммита синхронной транзакции. Как только это произойдет, асинхронная транзакция сразу сможет пройти коммит, не ожидая репликации.

**Предупреждение:** Будьте осторожны при одновременном использовании синхронных и асинхронных транзакций. Для асинхронных транзакций коммит засчитывается успешным даже в том случае, если нет соединения с другими узлами. Поэтому на старом `leader`-узле (*владельце очереди синхронных транзакций*) могут быть асинхронные транзакции, которые прошли коммит, но отсутствуют на других узлах набора реплик.

Когда предыдущий лидер снова становится доступен в наборе реплик, он начинает получать данные с нового лидера. В это же время остальные узлы в наборе реплик начинают получать данные со

старого лидера, которых у них еще нет. Эти данные содержат и асинхронные транзакции, прошедшие коммит. В этот момент система выдаст ошибку *ER_SPLIT_BRAIN*, заставляя пользователя провести повторную настройку (rebootstrap) предыдущего лидера.

### Ограничения и известные проблемы

До версии *2.5.2* способа настроить синхронную репликацию для существующих спейсов не было. Однако, начиная с версии 2.5.2, ее можно включить, вызвав метод *space_object:alter({is_sync = true})*.

Синхронные транзакции работают исключительно в топологии «мастер-реплика». В кластере может быть несколько реплик, в том числе анонимных, однако синхронные транзакции должен совершать только один узел.

Начиная с версии Tarantool *2.10.0*, анонимные реплики не участвуют в кворуме.

### Выборы лидера

В Tarantool, начиная с версии *2.6.1*, есть встроенная функциональность для управления автоматическими выборами лидера (automated leader election) в наборе реплик. Подробности можно найти в *соответствующей главе*.

### Автоматические выборы лидера

В Tarantool, начиная с версии *2.6.1*, есть встроенная функциональность для управления автоматическими *выборами лидера* (automated leader election) в наборе реплик (replica set). Эта функциональность повышает отказоустойчивость систем на базе Tarantool и снижает зависимость от внешних инструментов для управления набором реплик.

To learn how to configure and monitor automated leader elections, check *Managing leader elections*.

Ниже описаны следующие темы:

- *Выборы лидера и синхронная репликация*
- *Процесс выборов лидера*
- *Managing leader elections*

### Выборы лидера и синхронная репликация

В Tarantool используется модификация Raft — алгоритма синхронной репликации и автоматических выборов лидера. Полное описание алгоритма Raft можно прочитать в соответствующем документе.

*Синхронная репликация* и выборы лидера в Tarantool реализованы как две независимые подсистемы. Это означает, что можно настроить синхронную репликацию, а для выборов лидера использовать альтернативный алгоритм. Встроенный механизм выборов лидера, в свою очередь, не требует использования синхронных спейсов. Синхронной репликации посвящен *этот раздел документации*. Процесс выборов лидера описан ниже.

---

**Примечание:** The system behavior can be specified exactly according to the Raft algorithm. To do this:

- Ensure that the user has only synchronous spaces.
- Set the *replication.synchro_quorum* option to `N / 2 + 1`.
- Set the *replication.synchro_timeout* option to infinity.
- In the *replication.election_fencing_mode* option, select either the `soft` mode (the default) or the `strict` mode, which is more restrictive.

### Процесс выборов лидера

Автоматические выборы лидера в Tarantool гарантируют, что в каждый момент времени в наборе реплик будет максимум один *лидер* — узел, доступный для записи. Все остальные узлы будут принимать исключительно запросы на чтение.

Когда *функция выборов включена*, жизненный цикл набора реплик разделен на так называемые *термы* (term). Каждый терм описывается монотонно растущим числом. После первой загрузки узла значение его терма равно 1. Когда узел обнаруживает, что не является лидером и при этом лидера в наборе реплик уже какое-то время нет, он увеличивает значение своего терма и начинает новый тур выборов.

Выборы лидера происходят посредством голосования. Узел, начинающий выборы, голосует сам за себя и отправляет другим запросы на голос. Каждый экземпляр голосует за первый узел, от которого пришел такой запрос, и далее в течение всего терма ожидает избрания лидера, не выполняя никаких действий.

The node that collected a quorum of votes defined by the *replication.synchro_quorum* parameter becomes the leader and notifies other nodes about that. Also, a split vote can happen when no nodes received a quorum of votes. In this case, after a random timeout, each node increases its term and starts a new election round if no new vote request with a greater term arrives during this time. Eventually, a leader is elected.

If any unfinalized synchronous transactions are left from the previous leader, the new leader finalizes them automatically.

All the non-leader nodes are called *followers*. The nodes that start a new election round are called *candidates*. The elected leader sends heartbeats to the non-leader nodes to let them know it is alive.

In case there are no heartbeats for the period of *replication.timeout* * 4, a non-leader node starts a new election if the following conditions are met:

- The node has a quorum of connections to other cluster members.
- None of these cluster members can see the leader node.

**Примечание:** A cluster member considers the leader node to be alive if the member received heartbeats from the leader at least once during the `replication.timeout * 4`, and there are no replication errors (the connection is not broken due to timeout or due to an error).

Terms and votes are persisted by each instance to preserve certain Raft guarantees.

При голосовании узлы отдают предпочтение экземплярам, где сохранены самые новые данные. Поэтому, если прежний лидер перед тем, как стать недоступным, отправит кворуму реплик какую-либо информацию, она не будет потеряна.

When election is enabled, there must be connections between each node pair so as it would be the full mesh topology. This is needed because election messages for voting and other internal things need a direct connection between the nodes.

In the classic Raft algorithm, a leader doesn't track its connectivity to the rest of the cluster. Once the leader is elected, it considers itself in the leader position until receiving a new term from another cluster node. This can lead to a split situation if the other nodes elect a new leader upon losing the connectivity to the previous one.

The issue is resolved in Tarantool version *2.10.0* by introducing the leader *fencing* mode. The mode can be switched by the *replication.election_fencing_mode* configuration parameter. When the fencing is set to `soft` or `strict`, the leader resigns its leadership if it has less than *replication.synchro_quorum* of alive connections to the cluster nodes. The resigning leader receives the status of a follower in the current election term and becomes read-only. Leader *fencing* can be turned off by setting the *replication.election_fencing_mode* configuration parameter to `off`.

In `soft` mode, a connection is considered dead if there are no responses for *4 * replication.timeout* seconds both on the current leader and the followers.

In `strict` mode, a connection is considered dead if there are no responses for *2 * replication.timeout* seconds on the current leader and for *4 * replication.timeout* seconds on the followers. This improves chances that there is only one leader at any time.

Fencing applies to the instances that have the *replication.election_mode* set to «candidate» or «manual».

There can still be a situation when a replica set has two leaders working independently (so-called *split-brain*). It can happen, for example, if a user mistakenly lowered the *replication.synchro_quorum* below `N / 2 + 1`. In this situation, to preserve the data integrity, if an instance detects the split-brain anomaly in the incoming replication data, it breaks the connection with the instance sending the data and writes the `ER_SPLIT_BRAIN` error in the log.

Eventually, there will be two sets of nodes with the diverged data, and any node from one set is disconnected from any node from the other set with the `ER_SPLIT_BRAIN` error.

Once noticing the error, a user can choose any representative from each of the sets and inspect the data on them. To correlate the data, the user should remove it from the nodes of one set, and reconnect them to the nodes from the other set that have the correct data.

Любой узел, участвующий в процессе выборов, реплицирует данные только с последнего избранного лидера. Это позволяет избежать ситуации, в которой прежний лидер после выборов нового все еще пытается отправлять изменения на реплики.

Числовые значения термов также выполняют функцию своеобразного фильтра. Например, если на двух узлах включена функция выборов и значение терма `node1` меньше значения терма `node2`, то узел `node2` не будет принимать транзакций от узла `node1`.

### Managing leader elections

### Configuration

```
replication:
  election_mode: <string>
  election_fencing_mode: <string>
  election_timeout: <seconds>
  timeout: <seconds>
  synchro_quorum: <count>
```

- *replication.election_mode* – specifies the role of a node in the leader election process.
- *replication.election_fencing_mode* – specifies the *leader fencing mode*.

- *replication.election_timeout* – specifies the timeout between election rounds if the previous round ended up with a split vote.
- *replication.timeout* – a time interval (in seconds) used by a master to send heartbeat requests to a replica when there are no updates to send to this replica.
- *replication.synchro_quorum* – a number of replicas that should confirm the receipt of a *synchronous* transaction before it can finish its commit.

It is important to know that being a leader is not the only requirement for a node to be writable. The leader should also satisfy the following requirements:

- The *database.mode* option is set to `rw`.
- The leader shouldn't be in the orphan state.

Nothing prevents you from setting the `database.mode` option to `ro`, but the leader won't be writable then. The option doesn't affect the election process itself, so a read-only instance can still vote and become a leader.

### Monitoring

To monitor the current state of a node regarding the leader election, use the *box.info.election* function.

**Example:**

```
tarantool> box.info.election
---
- state: follower
  vote: 0
  leader: 0
  term: 1
...
```

The Raft-based election implementation logs all its actions with the `RAFT:` prefix. The actions are new Raft message handling, node state changing, voting, and term bumping.

### Important notes

Leader election doesn't work correctly if the election quorum is set to less or equal than `<cluster size> / 2`. In that case, a split vote can lead to a state when two leaders are elected at once.

For example, suppose there are five nodes. When the quorum is set to `2`, `node1` and `node2` can both vote for `node1`. `node3` and `node4` can both vote for `node5`. In this case, `node1` and `node5` both win the election. When the quorum is set to the cluster majority, that is `(<cluster size> / 2) + 1` or greater, the split vote is impossible.

That should be considered when adding new nodes. If the majority value is changing, it's better to update the quorum on all the existing nodes before adding a new one.

Also, the automated leader election doesn't bring many benefits in terms of data safety when used *without synchronous replication*. If the replication is asynchronous and a new leader gets elected, the old leader is still active and considers itself the leader. In such case, nothing stops it from accepting requests from clients and making transactions. Non-synchronous transactions are successfully committed because they are not checked against the quorum of replicas. Synchronous transactions fail because they are not able to collect the quorum – most of the replicas reject these old leader's transactions since it is not a leader anymore.

For practical guides to replication, see *Replication tutorials*. You can learn about bootstrapping a replica set, adding instances to the replica set, or removing them.

### 4.8.8 Триггеры

**Триггеры**, которые также называют **обратными вызовами**, представляют собой функции, которые выполняет сервер при наступлении определенных событий.

Чтобы связать событие с колбэк-функцией, передайте её в соответствующую функцию обработки событий `on_event`:

- *box.session.on_connect()* или *box.session.on_disconnect()*,
- *box.session.on_auth()*,
- *space_object:on_replace()* или *space_object:before_replace()*,
- *box.on_commit()* или *box.on_rollback()*,
- *net.box.on_connect()* или *net.box.on_disconnect()*,
- *net.box.on_schema_reload()*,
- *box.ctl.on_schema_init()* или *box.ctl.on_shutdown()*,
- *swim_object:on_member_event()*,
- *box.session.on_access_denied()*.

Тогда сервер сохранит колбэк-функцию и будет вызывать ее при наступлении соответствующего события.

У всех триггеров есть следующие особенности:

- Только *пользователь «admin»* определяет триггеры.
- Триггеры хранятся в памяти экземпляра Tarantool, а не в базе данных. Поэтому триггеры пропадают, когда экземпляр отключают. Чтобы сохранить их, поместите определения функции и настройки триггера в *скрипт инициализации* Tarantool.
- Триггеры не тратят много ресурсов. Если триггер не задан, то требуется минимум вычислений — разыменование и проверка указателя. Если триггер определен, то стоимость вызова равна стоимости вызова функции.
- Для одного события можно определить несколько триггеров. В таком случае триггеры выполняются в обратном порядке относительно того, как их определили.
- Триггеры должны работать в контексте события, то есть оперировать переменными, которые переданы как аргументы его функции. Триггеры не должны влиять на глобальное состояние программы или менять параметры, не связанные с событием. Если триггер выполняет такие вызовы, как, например, os.exit() или *box.rollback()*, результат его выполнения не определен.
- Триггеры можно заменять. Запрос на «замену триггера» подразумевает передачу новой функции-триггера и старой функции-триггера в одну из функций обработки событий `on_event`.
- Во всех функциях обработки событий `on_event` есть параметры, которые представляют собой указатели функции, и все они возвращают указатели функции. Следует запомнить, что определение Lua-функции, например, `function f() x = x + 1 end` значит то же самое, что `f = function () x = x + 1 end`, поскольку в обоих случаях `f` получит указатель функции. А `trigger = box.session.on_connect(f)` значит то же самое, что `trigger = box.session.on_connect(function () x = x + 1 end)` — в обоих случаях `trigger` получит переданный указатель функции.

- Если вызвать любую из функций обработки событий `on_event` без аргументов, то она вернет список соответствующих триггеров. Например, `box.session.on_connect()` вернет таблицу со всеми функциями-триггерами на подключение.
- Триггеры можно использовать для решения проблем с репликацией. Более подробную информацию см. в разделе *Решение конфликтов репликации*.

**Пример:**

Здесь мы записываем события подключения и отключения в журнал на сервере Tarantool.

```
log = require('log')

function on_connect_impl()
 log.info("connected "..box.session.peer().."", sid "..box.session.id())
end

function on_disconnect_impl()
 log.info("disconnected, sid "..box.session.id())
end

function on_auth_impl(user)
 log.info("authenticated sid "..box.session.id().." as "..user)
end

function on_connect() pcall(on_connect_impl) end
function on_disconnect() pcall(on_disconnect_impl) end
function on_auth(user) pcall(on_auth_impl, user) end

box.session.on_connect(on_connect)
box.session.on_disconnect(on_disconnect)
box.session.on_auth(on_auth)
```

### 4.8.9 Движки базы данных

Движок базы данных — это набор низкоуровневых процессов, которые фактически хранят и получают значения *кортежей*. Tarantool предлагает выбор из двух движков базы данных:

- *memtx* — in-memory движок базы данных, используемый по умолчанию.
- *vinyl* — дисковый движок базы данных.

Все подробности о том, как работают движки, вы можете найти в следующих разделах:

#### Хранение данных с помощью memtx

The `memtx` storage engine is used in Tarantool by default. The engine keeps all data in random-access memory (RAM), and therefore has a low read latency.

Tarantool prevents the data loss in case of emergency, such as outage or Tarantool instance failure, in the following ways:

- Tarantool persists all data changes by writing requests to the *write-ahead log* (WAL) that is stored on disk. Also, Tarantool periodically takes the entire *database snapshot* and saves it on disk. Learn more: *Data persistence*.

- In case of a distributed application, a synchronous replication is used to ensure keeping the data consistent on a quorum of replicas. Although replication is not directly a storage engine topic, it is a part of the answer regarding data safety. Learn more: *Replicating data*.

In this section, the following topics are discussed in brief with the references to other sections that explain the subject matter in details.


- *Модель памяти*
- *Персистентность данных*
- *Доступ к данным*
- *Репликация данных*
- *Ключевые сведения*


### Модель памяти

There is a fixed number of independent *execution threads*. The threads don't share state. Instead they exchange data using low-overhead message queues. While this approach limits the number of cores that the instance uses, it removes competition for the memory bus and ensures peak scalability of memory access and network throughput.

Only one thread, namely, the **transaction processor thread** (further, **TX thread**) can access the database, and there is only one TX thread for each Tarantool instance. In this thread, transactions are executed in a strictly consecutive order. Multi-statement transactions exist to provide isolation: each transaction sees a consistent database state and commits all its changes atomically. At commit time, a yield happens and all transaction changes are written to *WAL* in a single batch. In case of errors during transaction execution, a transaction is rolled-back completely. Read more in the following sections: *Transaction model*, *Transaction mode: MVCC*.

Внутри потока TX есть область памяти, в которой Tarantool хранит данные. Эта область называется **Arena**.

Data is stored in *spaces*. Spaces contain database records – *tuples*. To access and manipulate the data stored in spaces and tuples, Tarantool builds *indexes*.

Распределением памяти для спейсов, кортежей и индексов внутри области Arena управляют специальные аллокаторы. Для хранения кортежей главным образом используется аллокатор slab. В Tarantool встроен модуль под названием `box.slab`, предоставляющий статистику распределения slab. С помощью этой статистики можно отслеживать общее использование памяти и ее фрагментацию. Подробности см. в *руководстве* по модулю `box.slab`.

Also inside the TX thread, there is an event loop. Within the event loop, there are a number of *fibers*. Fibers are cooperative primitives that allow interaction with spaces, that is, reading and writing the data. Fibers can interact with the event loop and between each other directly or by using special primitives called channels. Due to the usage of fibers and *cooperative multitasking*, the `memtx` engine is lock-free in typical situations.

To interact with external users, there is a separate *network thread* also called the **iproto thread**. The iproto thread receives a request from the network, parses and checks the statement, and transforms it into a special

structure—a message containing an executable statement and its options. Then the iproto thread ships this message to the TX thread and runs the user's request in a separate fiber.

### Персистентность данных

Tarantool ensures *data persistence* as follows:

- Исполнив в памяти запросы на изменение данных, Tarantool записывает все эти запросы в файлы *журнала упреждающей записи (WAL)* (с расширением `.xlog`), хранящиеся на диске. Это делается в отдельном потоке, называемом **поток WAL**.
- Tarantool периодически делает *снимок всей базы данных* и сохраняет его на диск. Это необходимо для ускорения перезапуска экземпляра, так как если файлов WAL слишком много, то Tarantool не сможет быстро перезапуститься.

  To save a snapshot, there is a special fiber called the *snapshot daemon*. It reads the consistent content of the entire Arena and writes it on disk into a snapshot file (`.snap`). Due of the cooperative multitasking, Tarantool cannot write directly on disk because it is a locking operation. That is why Tarantool interacts with disk via a separate pool of threads from the *fio* library.

Таким образом, при перезапуске Tarantool данные можно полностью восстановить даже в аварийных ситуациях, например при отключении питания или падении экземпляра Tarantool, когда хранящаяся в оперативной памяти база данных утеряна.

Что происходит при перезапуске:

1. Tarantool находит и читает последний файл снимка.
2. Tarantool также находит и читает все файлы WAL, созданные после этого снимка.
3. Как только снимок и файлы WAL будут прочитаны, набор данных в памяти будет полностью восстановлен. Он будет соответствовать состоянию экземпляра Tarantool на момент, когда тот прекратил работу.
4. Во время чтения снимка и файлов WAL Tarantool строит первичные индексы.
5. Когда все данные снова в памяти, Tarantool строит вторичные индексы.
6. Tarantool запускает приложение.

### Доступ к данным

Чтобы обращаться к данным, хранящимся в оперативной памяти, и работать с ними, Tarantool строит индексы, которые хранятся внутри области памяти Arena.

Tarantool поддерживает несколько *типов индексов*: TREE, HASH, BITSET, RTREE. Все они предназначены для разных сценариев использования.

Можно выполнять SELECT-запросы как по первичным, так и по вторичным ключам индекса. Ключи могут быть составными.

Подробности про индексы можно найти на странице *Индексы*.

### Репликация данных

Хотя эта тема не имеет прямого отношения к движку `memtx`, она дополняет общую картину того, как работает Tarantool, когда приложение распределенное.

Репликация позволяет нескольким экземплярам Tarantool работать с копиями одной и той же базы данных. Эти копии остаются синхронизированными благодаря тому, что каждый экземпляр может сообщать другим экземплярам о совершенных им изменениях. Для этого используется WAL-репликация.

Чтобы отправить данные на реплику, Tarantool запускает еще один поток, называемый **relay**. Этот поток читает файлы WAL и отправляет их репликам. На каждой реплике выполняется файбер под названием **applier**. Он получает изменения от удаленного узла и применяет их к области Arena реплики. Все изменения записываются в файлы WAL через поток WAL реплики так же, как если бы они были сделаны локально.

В Tarantool *репликация* по умолчанию асинхронна: то, что транзакция проходит коммит локально на главном узле, не означает, что она отправляется на какие-то другие реплики.

Эту проблему решает *синхронная репликация*. Каждая синхронная транзакция проходит коммит лишь после репликации на некотором количестве экземпляров, и только тогда клиенту приходит ответ о завершении транзакции.

Более подробные сведения вы найдете в *главе о репликации*.

### Ключевые сведения

Вот главные принципы, по которым работает движок:

- Все данные находятся в оперативной памяти.
- Доступ к данным производится только из одного потока.
- Tarantool записывает все изменения данных в файлы WAL.
- Периодически создаются снимки данных.
- Для доступа к данным создаются индексы.
- Файлы WAL можно реплицировать.

### Хранение данных с помощью vinyl

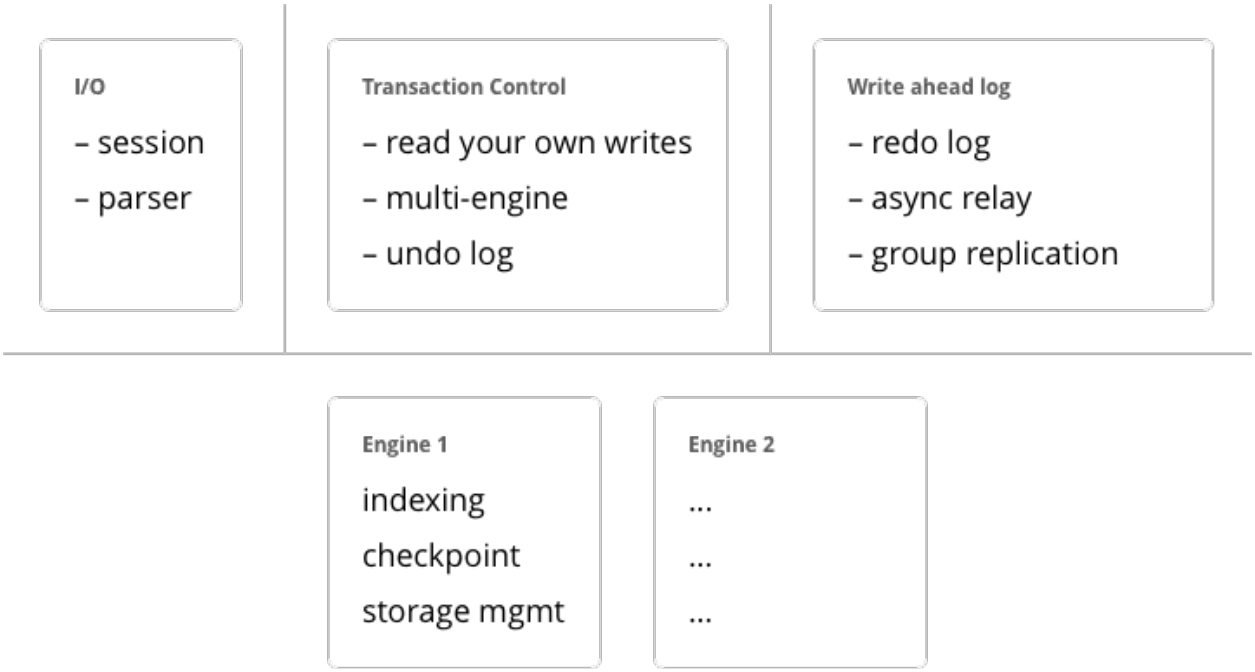
Tarantool – это транзакционная, персистентная СУБД, которая хранит 100% данных в оперативной памяти. Основными преимущества хранения данных оперативной памяти являются скорость и простота использования: нет необходимости в оптимизации, однако производительность остается стабильно высокой.

Несколько лет назад мы решили расширить продукт путем реализации классической технологии хранения как в обычных СУБД: в оперативной памяти хранится лишь кэш данных, а основной объем данных находится на диске. Мы решили, что движок хранения можно будет выбирать независимо для каждой таблицы, как это реализовано в MySQL, но при этом с самого начала будет реализована поддержка транзакций.

Первый вопрос, на который нужен был ответ: создавать свой движок или использовать уже существующую библиотеку? Сообщество разработчиков открытого ПО предлагает готовые библиотеки на

выбор. Активнее всего развивалась библиотека RocksDB, которая к настоящему времени стала одной из самых популярных. Есть также несколько менее известных библиотек: WiredTiger, ForestDB, NestDB, LMDB.

Тем не менее, изучив исходный код существующих библиотек и взвесив все «за» и «против», мы решили написать свой движок. Одна из причин – все существующие сторонние библиотеки предполагают, что запросы к данным могут поступать из множества потоков операционной системы, и поэтому содержат сложные примитивы синхронизации для управления одновременным доступом к данным. Если бы мы решили встраивать одну из них в Tarantool, то пользователи были бы вынуждены нести издержки многопоточных приложений, не получая ничего взамен. Дело в том, что в основе Tarantool лежит архитектура на основе акторов. Обработка транзакций в выделенном потоке позволяет обойтись без лишних блокировок, межпроцессного взаимодействия и других затрат ресурсов, которые забирают до 80% процессорного времени в многопоточных СУБД.




*Процесс в Tarantool состоит из заданного количества потоков*

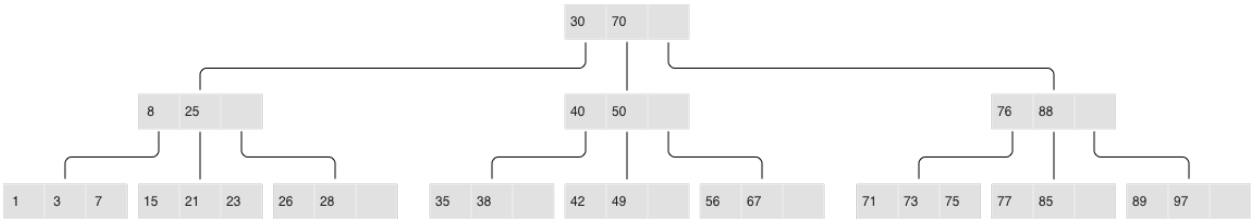
Если изначально проектировать движок с учетом кооперативной многозадачности, можно не только существенно ускорить работу, но и реализовать приемы оптимизации, слишком сложные для многопоточных движков. В общем, использование стороннего решения не привело бы к лучшему результату.

### Алгоритм

Отказавшись от идеи внедрения существующих библиотек, необходимо было выбрать архитектуру для использования в качестве основы. Есть два альтернативных подхода к хранению данных на диске: старая модель с использованием B-деревьев и их разновидностей и новая – на основе журнально-структурированных деревьев со слиянием, или LSM-деревьев (Log Structured Merge Tree). MySQL, PostgreSQL и Oracle используют B-деревья, а Cassandra, MongoDB и CockroachDB уже используют LSM-деревья.

Считается, что B-деревья более эффективны для чтения, а LSM-деревья – для записи. Тем не менее, с распространением SSD-дисков, у которых в несколько раз выше производительность чтения по сравнению с производительностью записи, преимущества LSM-деревьев стали очевидны в большинстве сценариев.

Прежде чем разбираться с LSM-деревьями в Tarantool, посмотрим, как они работают. Для этого разберем устройство обычного B-дерева и связанные с ним проблемы. «B» в слове B-tree означает «Block», то есть это сбалансированное дерево, состоящее из блоков, которые содержат отсортированные списки пар ключ-значение. Вопросы наполнения дерева, балансировки, разбиения и слияния блоков выходят за рамки данной статьи, подробности вы сможете прочитать в Википедии. В итоге мы получаем отсортированный по возрастанию ключа контейнер, минимальный элемент которого хранится в крайнем левом узле, а максимальный – в крайнем правом. Посмотрим, как в B-дереве осуществляется поиск и вставка данных.




*Классическое B-дерево*

Если необходимо найти элемент или проверить его наличие, поиск начинается, как обычно, с вершины. Если ключ обнаружен в корневом блоке, поиск заканчивается; в противном случае, переходим в блок с наибольшим меньшим ключом, то есть в самый правый блок, в котором еще есть элементы меньше искомого (элементы на всех уровнях расположены по возрастанию). Если и там элемент не найден, снова переходим на уровень ниже. В конце концов окажемся в одном из листьев и, возможно, обнаружим искомый элемент. Блоки дерева хранятся на диске и читаются в оперативную память по одному, то есть в рамках одного поиска алгоритм считывает $logB(N)$ блоков, где N – это количество элементов в B-дереве. Запись в самом простом случае осуществляется аналогично: алгоритм находит блок, который содержит необходимый элемент, и обновляет (вставляет) его значение.

Чтобы наглядно представить себе эту структуру данных, возьмем B-дерево на 100 000 000 узлов и предположим, что размер блока равен 4096 байтов, а размер элемента – 100 байтов. Таким образом, в каждом блоке можно будет разместить до 40 элементов с учетом накладных расходов, а в B-дереве будет около 2 570 000 блоков, пять уровней, при этом первые четыре займут по 256 МБ, а последний – до 10 ГБ. Очевидно, что на любом современном компьютере все уровни, кроме последнего, успешно попадут в кэш файловой системы, и фактически любая операция чтения будет требовать не более одной операции ввода-вывода.

Ситуация выглядит существенно менее радужно при смене точки зрения. Предположим, что необходимо обновить один элемент дерева. Так как операции с B-деревьями работают через чтение и запись целых блоков, приходится прочитать 1 блок в память, изменить 100 байт из 4096, а затем записать обновленный блок на диск. Таким образом, нам пришлось записать в 40 раз больше, чем реальный объем измененных данных!

Принимая во внимание, что внутренний размер блока в SSD-дисках может быть 64 КБ и больше, и не любое изменение элемента меняет его целиком, объем «паразитной» нагрузки на диск может быть еще выше.

Феномен таких «паразитных» чтений в литературе и блогах, посвященных хранению на диске, называется read amplification (усложнение чтения), а феномен «паразитной» записи – write amplification (усложнение записи).

Коэффициент усложнения, то есть коэффициент умножения, вычисляется как отношение размера фактически прочитанных (или записанных) данных к реально необходимому (или измененному) размеру. В нашем примере с B-деревом коэффициент составит около 40 как для чтения, так и для записи.

Объем «паразитных» операций ввода-вывода при обновлении данных является одной из основных проблем, которую решают LSM-деревья. Рассмотрим, как это работает.

Ключевое отличие LSM-деревьев от классических B-деревьев заключается в том, что LSM-деревья не просто хранят данные (ключи и значения), а также операции с данными: вставки и удаления.

| key | lsn | op_code | value |
|---|---|---|---|

LSM-дерево:

- **Хранит операторы, а не значения:**
  - REPLACE
  - DELETE
  - UPSERT
- Для каждого оператора назначается LSN
- Обновление файлов происходит только путем присоединения новых записей, сборка мусора проводится после контрольной точки
- Журнал транзакций при любых изменениях в системе: vylog

Например, элемент для операции вставки, помимо ключа и значения, содержит дополнительный байт с кодом операции – обозначенный выше как REPLACE. Элемент для операции удаления содержит ключ элемента (хранить значение нет необходимости) и соответствующий код операции – DELETE. Также каждый элемент LSM-дерева содержит порядковый номер операции (log sequence number – LSN), то есть значение монотонно возрастающей последовательности, которое уникально идентифицирует каждую операцию. Таким образом, всё дерево упорядочено сначала по возрастанию ключа, а в пределах одного ключа – по убыванию LSN.

| Key | lsn | Op code | Value |
|---|---|---|---|
| 1 | 176 | REPLACE | 2018-05-07 15:00:01 |
| 1 | 53 | INSERT | 2017-12-31 23:59:01 |
| 2 | 174 | REPLACE | 2018-05-06 00:00:00 |
| 3 | 175 | REPLACE | 2018-05-07 09:04:19 |
| 3 | 9 | REPLACE | 2017-01-01 19:25:43 |
| 3 | 7 | INSERT | 2017-01-01 19:22:16 |
| 4 | 173 | DELETE | |
| 4 | 168 | INSERT | 2018-05-05 07:40:01 |

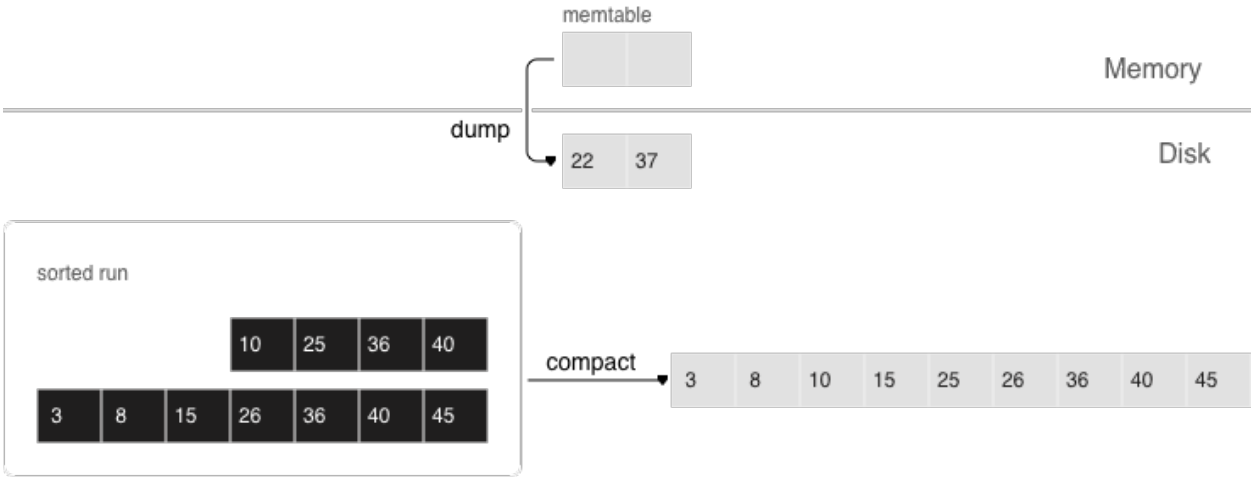
*Один уровень LSM-дерева*

### Наполнение LSM-дерева

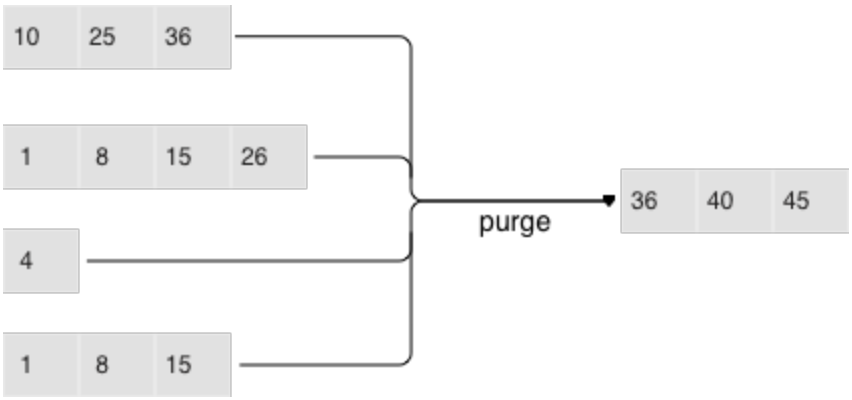
В отличие от B-дерева, которое полностью хранится на диске и может частично кэшироваться в оперативной памяти, в LSM-дереве разделение между памятью и диском явно присутствует с самого начала. При этом проблема сохранности данных, расположенных в энергозависимой памяти, выносится за рамки алгоритма хранения: ее можно решить разными способами, например, журналированием изменений.

Часть дерева, расположенную в оперативной памяти, называют L0 (level zero – уровень ноль). Объем оперативной памяти ограничен, поэтому для L0 отводится фиксированная область. В конфигурации Tarantool, например, размер L0 задается с помощью параметра `vinyl_memory`. В начале, когда LSM-дерево не содержит элементов, операции записываются в L0. Следует отметить, что элементы в дереве упорядочены по возрастанию ключа, а затем по убыванию LSN, так что в случае вставки нового значения по данному ключу легко обнаружить и удалить предыдущее значение. L0 может быть представлен любым контейнером, который сохраняет упорядоченность элементов. Например, для хранения L0 Tarantool использует B+*-дерево. Операции поиска и вставки – это стандартные операции структуры данных, используемой для представления L0, и мы их подробно рассматривать не будем.

Рано или поздно количество элементов в дереве превысит размер L0. Тогда L0 записывается в файл на диске (который называется забегом – «run») и освобождается под новые элементы. Эта операция называется «дамп» (dump).




Все дампы на диске образуют последовательность, упорядоченную по LSN: диапазоны LSN в файлах не пересекаются, а ближе к началу последовательности находятся файлы с более новыми операциями. Представим эти файлы в виде пирамиды, где новые файлы расположены вверху, а старые внизу. По мере появления новых файлов забегов, высота пирамиды растет. При этом более свежие файлы могут содержать операции удаления или замены для существующих ключей. Для удаления старых данных необходимо производиться сборку мусора (этот процесс иногда называется «слияние» – в английском языке «merge» или «compaction»), объединяя нескольких старых файлов в новый. Если при слиянии мы встречаем две версии одного и того же ключа, то достаточно оставить только более новую версию, а если после вставки ключа он был удален, то из результата можно исключить обе операции.




Ключевым фактором эффективности LSM-дерева является то, в какой момент и для каких файлов делается слияние. Представим, что LSM-дерево в качестве ключей хранит монотонную последовательность вида 1, 2, 3 . . . , и операций удаления нет. В этом случае слияние будет бесполезным – все элементы уже отсортированы, дерево не содержит мусор и можно однозначно определить, в каком файле находится каждый ключ. Напротив, если LSM-дерево содержит много операций удаления, слияние позволит освободить место на диске. Но даже если удалений нет, а диапазоны ключей в разных файлах сильно пересекаются, слияние может ускорить поиск, так как сократит число просматриваемых файлов. В этом случае имеет смысл выполнять слияние после каждого дампа. Однако следует отметить, что такое слияние приведет к перезаписи всех данных на диске, поэтому если чтений мало, то лучше делать слияния реже.

Для оптимальной конфигурации под любой из описанных выше сценариев в LSM-дереве все файлы организованы в пирамиду: чем новее операции с данными, тем выше они находятся в пирамиде. При этом в слиянии участвуют два или несколько соседних файлов в пирамиде; по возможности выбираются файлы примерно одинакового размера.

- Многоуровневое слияние может охватить любое количество уровней
- Уровень может содержать несколько файлов

Все соседние файлы примерно одинакового размера составляют уровень LSM-дерева на диске. Соотношение размеров файлов на различных уровнях определяет пропорции пирамиды, что позволяет оптимизировать дерево под интенсивные вставки, либо интенсивные чтения.

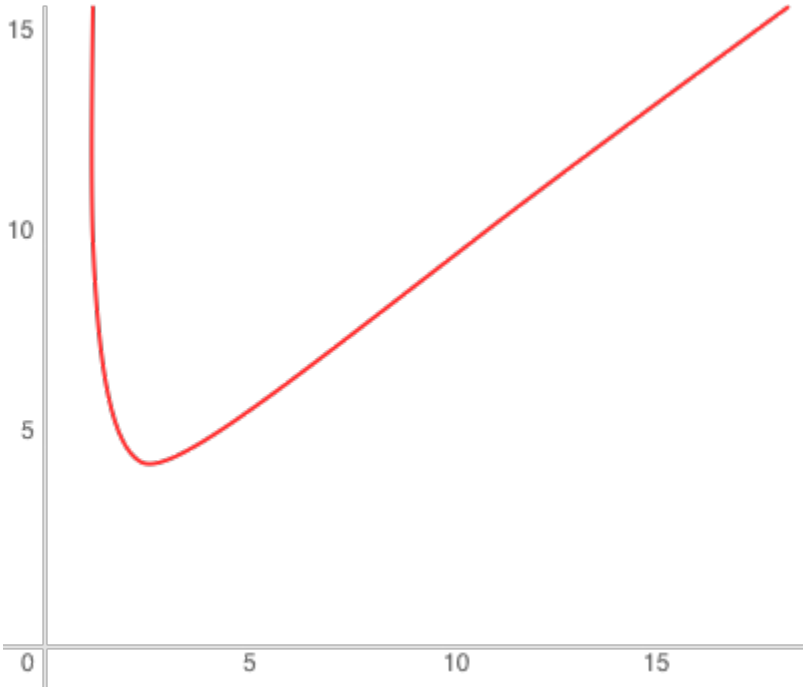
Предположим, что размер L0 составляет 100 МБ, а соотношение размеров файлов на каждом уровне (параметр `vinyl_run_size_ratio`) равно 5, и на каждом уровне может быть не более 2 файлов (параметр `vinyl_run_count_per_level`). После первых трех дампов на диске появятся 3 файла по 100 МБ,

эти файлы образуют уровень L1. Так как $3 > 2$, запустится слияние файлов в новый файл размером 300 МБ, а старые будут удалены. Спустя еще 2 дампа снова запустится слияние, на этот раз файлов в 100, 100 и 300 МБ, в результате файл размером 500 МБ переместится на уровень L2 (вспомним, что соотношение размеров уровней равно 5), а уровень L1 останется пустым. Пройдут еще 10 дампов, и получим 3 файла по 500 МБ на уровне L2, в результате чего будет создан один файл размером 1500 МБ. Спустя еще 10 дампов произойдет следующее: 2 раза произведем слияние 3 файлов по 100 МБ, а также 2 раза слияние файлов по 100, 100 и 300 МБ, что приведет к созданию двух файлов на уровне L2 по 500 МБ. Поскольку на уровне L2 уже есть три файла, запустится слияние двух файлов по 500 МБ и одного файла в 1500 МБ. Полученный в результате файл в 2500 МБ, в силу своего размера, переедет на уровень L3.

Процесс может продолжаться до бесконечности, а если в потоке операций с LSM-деревом будет много удалений, образовавшийся в результате слияния файл может переместиться не только вниз по пирамиде, но и вверх, так как окажется меньше исходных файлов, использовавшихся при слиянии. Иными словами, принадлежность файла к уровню достаточно отслеживать логически на основе размера файла и минимального и максимального LSN среди всех хранящихся в нем операций.

### Управление формой LSM-дерева

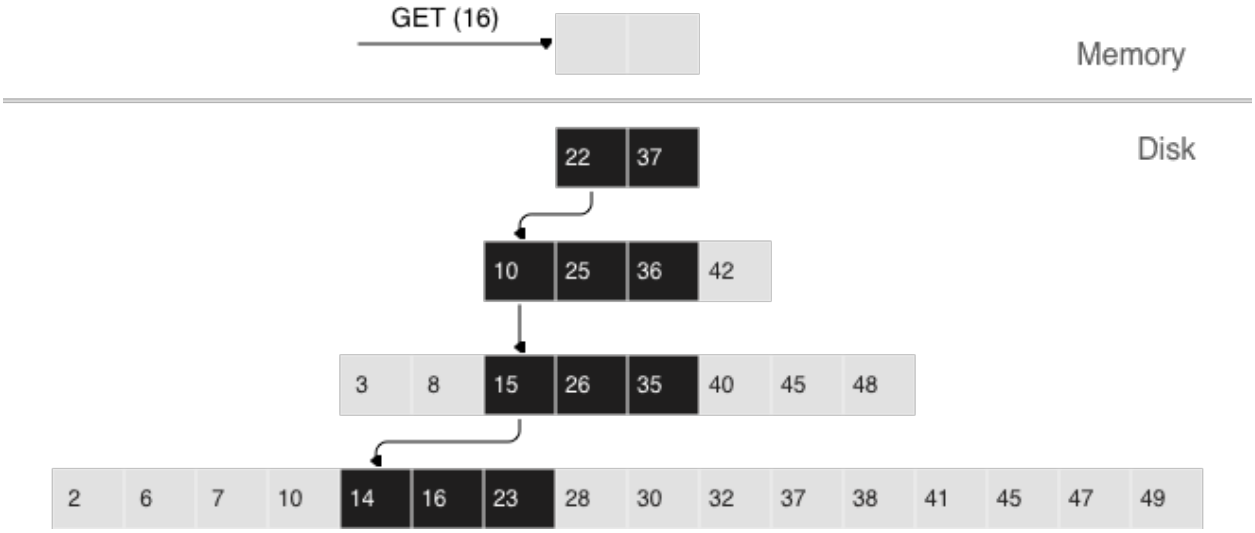
Если число файлов для поиска нужно уменьшить, то соотношение размеров файлов на разных уровнях можно увеличить, и, как следствие, уменьшается число уровней. Если, напротив, необходимо снизить затраты ресурсов, вызванные слиянием, то можно уменьшить соотношение размеров уровней: пирамида будет более высокой, а слияние хотя и выполняется чаще, но работает в среднем с файлами меньшего размера, за счет чего суммарно выполняет меньше работы. В целом, «паразитная запись» в LSM-дереве описывается формулой $log_x($
$fracNL0)x$ или $x$
$fracln(fracNC0)ln(x)$, где N – это общий размер всех элементов дерева, L0 – это размер уровня ноль, а x – это соотношение размеров уровней (параметр `level_size_ratio`). Если
$fracNC0 = 40$ (соотношение диск-память), график выглядит примерно вот так:



«Паразитное» чтение при этом пропорционально количеству уровней. Стоимость поиска на каждом уровне не превышает стоимости поиска в B-дереве. Возвращаясь к нашему примеру дерева в 100 000 000 элементов: при наличии 256 МБ оперативной памяти и стандартных значений параметров `vinyl_run_size_ratio` и `vinyl_run_count_per_level`, получим коэффициент «паразитной» записи равным примерно 13, коэффициент «паразитной» записи может доходить до 150. Разберемся, почему это происходит.

### Поиск

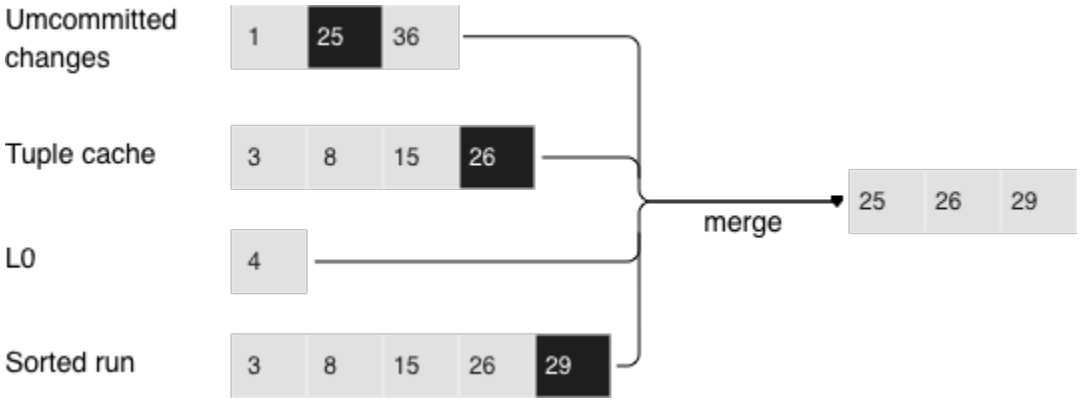
При поиске в LSM-дереве нам необходимо найти не сам элемент, а последнюю операцию с ним. Если это операция удаления, искомый элемент отсутствует в дереве. Если это операция вставки, то искомому элементу соответствует самое верхнее значение в LSM-пирамиде, и поиск можно остановить при первом совпадении ключа. В худшем случае значение в дереве изначально отсутствовало. Тогда поиск вынужден последовательно перебрать все уровни дерева, начиная с L0.




К сожалению, на практике этот худший случай довольно распространен. Например, при вставке в дерево необходимо убедиться в отсутствии дубликатов для первичного или уникального ключа. Поэтому для ускорения поиска несуществующих значений в LSM-деревьях применяется вероятностная структура данных, которая называется «фильтр Блума». О нем мы поговорим более детально в разделе, посвященном внутреннему устройству vinyl.

### Поиск по диапазону

Если при поиске по одному ключу алгоритм завершается после первого совпадения, то для поиска всех значений в диапазоне (например, всех пользователей с фамилией «Иванов») необходимо просматривать все уровни дерева.




*Поиск по диапазону [24,30)*

Формирование искомого диапазона при этом происходит так же, как и при слиянии нескольких файлов: из всех источников алгоритм выбирает ключ с максимальным LSN, отбрасывает остальные операции по этому ключу, сдвигает позицию поиска на следующий ключ и повторяет процедуру.

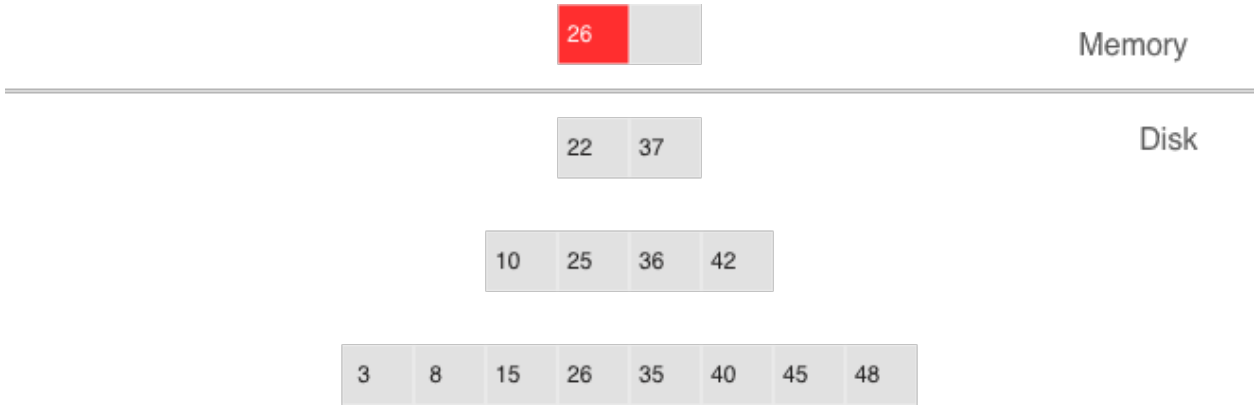
### Удаление

Зачем вообще хранить операции удаления? И почему это не приводит к переполнению дерева, например, в сценарии for i=1,10000000 put(i) delete(i) end?

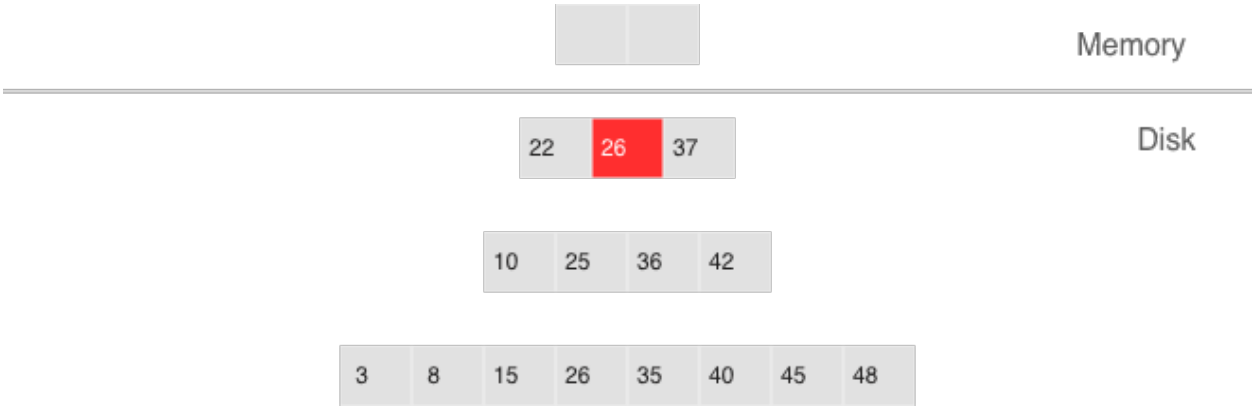
Роль операций удаления при поиске – сообщать об отсутствии искомого значения, а при слиянии – очищать дерево от «мусорных» записей с более старыми LSN.

Пока данные хранятся только в оперативной памяти, нет необходимости хранить операции удаления. Также нет необходимости сохранять операции удаления после слияния, если оно затрагивает в том числе самый нижний уровень дерева – на нем находятся данные самого старого дампа. Действительно, отсутствие значения на последнем уровне означает, что оно отсутствует в дереве.

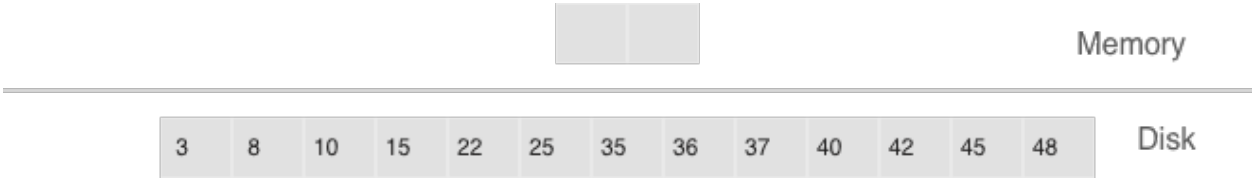
- Нельзя производить удаление из файлов, которые обновляются только путем присоединения новых записей
- Вместо этого на уровень L0 вносятся маркеры удаленных записей (tombstones)



*Удаление, шаг 1: вставка удаленной записи в L0*




*Удаление, шаг 2: удаленная запись проходит через промежуточные уровни*




*Удаление, шаг 3: при значительном слиянии удаленная запись удаляется из дерева*

Если мы знаем, что удаление следует сразу за вставкой уникального значения – а это частый случай при изменении значения во вторичном индексе – то операцию удаления можно отфильтровывать уже при слиянии промежуточных уровней. Эта оптимизация реализована в vinyl'е.

### Преимущества LSM-дерева

Помимо снижения «паразитной» записи, подход с периодическими дампами уровня L0 и слиянием уровней L1-Lk имеет ряд преимуществ перед подходом к записи, используемым в B-деревьях:

- При дампах и слиянии создаются относительно большие файлы: стандартный размер L0 составляет 50-100 МБ, что в тысячи раз превышает размер блока B-дерева.
- Большой размер позволяет эффективно сжимать данные перед записью. В Tarantool сжатие происходит автоматически, что позволяет еще больше уменьшить «паразитную» запись.
- Издержки фрагментации отсутствуют, потому что в файле элементы следуют друг за другом без пустот/заполнений.
- Все операции создают новые файлы, а не заменяют старые данные. Это позволяет избавиться от столь ненавистных нам блокировок, при этом несколько операций могут идти параллельно, не приводя к конфликтам. Это также упрощает создание резервных копий и перенос данных на реплику.
- Хранение старых версий данных позволяет эффективно реализовать поддержку транзакций, используя подход управления параллельным доступом с помощью многоверсионности.

### Недостатки LSM-дерева и их устранение

Одним из ключевых преимуществ B-дерева как структуры данных для поиска является предсказуемость: любая операция занимает не более чем $log_B(N)$. В классическом LSM-дереве скорость как чтения, так и записи могут может отличаться в лучшем и худшем случае в сотни и тысячи раз. Например, добавление всего лишь одного элемента в L0 может привести к его переполнению, что в свою очередь, может привести к переполнению L1, L2 и т.д. Процесс чтения может обнаружить исходный элемент в L0, а может задействовать все уровни. Чтение в пределах одного уровня также необходимо оптимизировать, чтобы добиться скорости, сравнимой с B-деревом. К счастью, многие недостатки можно скрасить или полностью устранить с помощью вспомогательных алгоритмов и структур данных. Систематизируем эти недостатки и опишем способы борьбы с ними, используемые в Tarantool.

### Непредсказуемая скорость записи

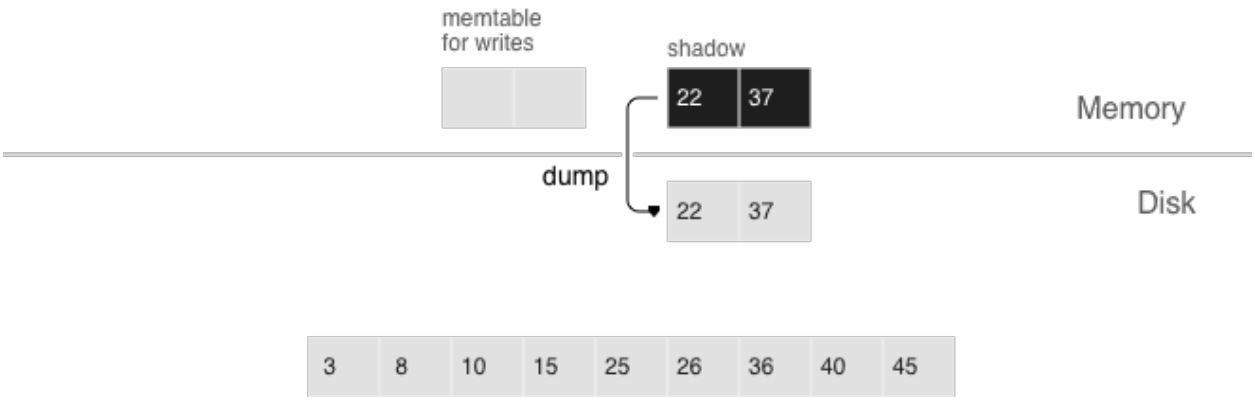
Вставка данных в LSM-дерево почти всегда задействует исключительно L0. Как избежать простоя, если заполнена область оперативной памяти, отведенная под L0?

Освобождение L0 подразумевает две долгих операции: запись на диск и освобождение памяти. Чтобы избежать простоя во время записи L0 на диск, Tarantool использует упреждающую запись. Допустим, размер L0 составляет 256 МБ. Скорость записи на диск составляет 10 МБ/с. Тогда для записи L0 на диск понадобится 26 секунд. Скорость вставки данных составляет 10 000 запросов в секунду, а размер одного ключа – 100 байтов. На время записи необходимо зарезервировать около 26 МБ доступной оперативной памяти, сократив реальный полезный размер L0 до 230 МБ.

Tarantool does all of these calculations automatically, constantly updating the rolling average of the DBMS workload and the histogram of the disk speed. This allows using L0 as efficiently as possible and it prevents write requests from timing out. But in the case of workload surges, some wait time is still possible. That's why we also introduced an insertion timeout (the `vinyl_timeout` parameter), which is set to 60 seconds by default. The write operation itself is executed in dedicated threads. The number of these threads (4 by default) is controlled by the `vinyl_write_threads` parameter. The default value of 2 allows doing dumps and compactions in parallel, which is also necessary for ensuring system predictability.

Слияния в Tarantool всегда выполняются независимо от дампов, в отдельном потоке выполнения. Это возможно благодаря природе LSM-дерева – после записи файлы в дереве никогда не меняются, а слияние лишь создает новый файл.

К задержкам также может приводить ротация L0 и освобождение памяти, записанной на диск: в процессе записи памятью L0 владеют два потока операционной системы – поток обработки транзакций и поток записи. Хотя в L0 во время ротации элементы не добавляются, он может участвовать в поиске. Чтобы избежать блокировок на чтение во время поиска, поток записи не освобождает записанную память, а оставляет эту задачу потоку обработки транзакций. Само освобождение после завершения дампа происходит мгновенно: для этого в L0 используется специализированный механизм распределения, позволяющий освободить всю память за одну операцию.

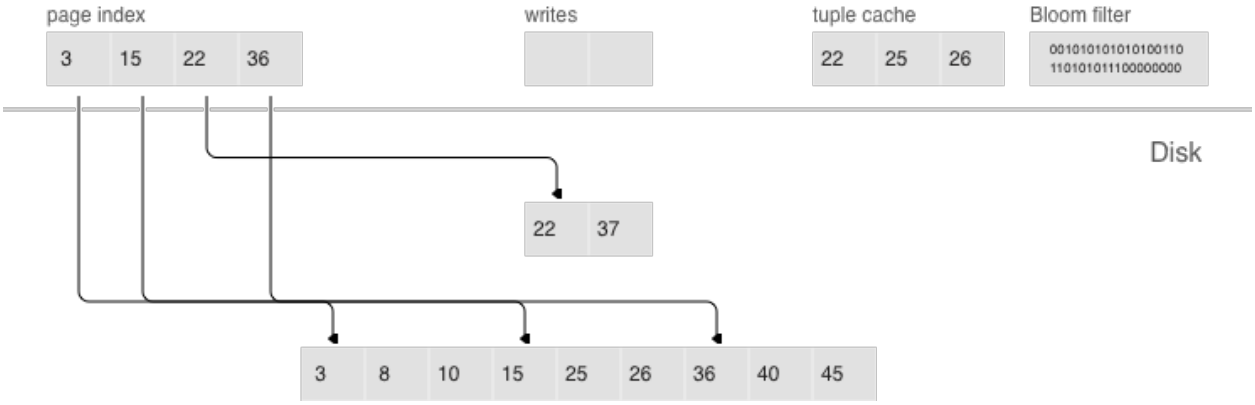



- упреждающий дамп
- загрузка

Дамп происходит из так называемого «теневого» L0, не блокируя новые вставки и чтения

### Непредсказуемая скорость чтений

Чтение – самая сложная задача для оптимизации в LSM-деревьях. Главным фактором сложности является большое количество уровней: это не только значительно замедляет поиск, но и потенциально значительно увеличивает требования к оперативной памяти при почти любых попытках оптимизации. К счастью, природа LSM-деревьев, где файлы обновляются только путем присоединения новых записей, позволяет решать эти проблемы нестандартными для традиционных структур данных способами.




- постраничный индекс
- фильтры Блума
- кэш диапазона кортежей
- многоуровневое слияние

### Сжатие и постраничный индекс

Сжатие данных в B-деревьях – это либо сложнейшая в реализации задача, либо больше средство маркетинга, чем действительно полезный инструмент. Сжатие в LSM-деревьях работает следующим образом:

При любом дампе или слиянии мы разбиваем все данные в одном файле на страницы. Размер страницы в байтах задается в параметре `vinyl_page_size`, который можно менять отдельно для каждого индекса. Страница не обязана занимать строго то количество байт, которое прописано `vinyl_page_size` – она может быть чуть больше или чуть меньше, в зависимости от хранящихся в ней данных. Благодаря этому страница никогда не содержит пустот.

Для сжатия используется потоковый алгоритм Facebook под названием «zstd». Первый ключ каждой страницы и смещение страницы в файле добавляются в так называемый постраничный индекс (page index) – отдельный файл, который позволяет быстро найти нужную страницу. После дампа или слияния постраничный индекс созданного файла также записывается на диск.

Все файлы типа `.index` кэшируются в оперативной памяти, что позволяет найти нужную страницу за одно чтение из файла `.run` (такое расширение имени файла используется в vinyl'е для файлов, полученных в результате дампа или слияния). Поскольку данные в странице отсортированы, после чтения и декомпрессии нужный ключ можно найти с помощью простого бинарного поиска. За чтение и декомпрессию отвечают отдельные потоки, их количество определяется в параметре `vinyl_read_threads`.

Tarantool использует единый формат файлов: например, формат данных в файле `.run` ничем не отличается от формата файла `.xlog` (файл журнала). Это упрощает резервное копирование и восстановление, а также работу внешних инструментов.

### Фильтры Блума

Хотя постраничный индекс позволяет уменьшить количество страниц, просматриваемых при поиске в одном файле, он не отменяет необходимости искать на всех уровнях дерева. Есть важный частный случай, когда необходимо проверить отсутствие данных, и тогда просмотр всех уровней неизбежен: вставка в уникальный индекс. Если данные уже существуют, то вставка в уникальный индекс должна завершиться с ошибкой. Единственный способ вернуть ошибку до завершения транзакции в LSM-дереве – произвести поиск перед вставкой. Такого рода чтения в СУБД образуют целый класс, называемый «скрытыми» или «паразитными» чтениями.

Другая операция, приводящая к скрытым чтениям, – обновление значения, по которому построен вторичный индекс. Вторичные ключи представляют собой обычные LSM-деревья, в которых данные хранятся в другом порядке. Чаще всего, чтобы не хранить все данные во всех индексах, значение, соответствующее данному ключу, целиком сохраняется только в первичном индексе (любой индекс, хранящий и ключ, и значение, называется покрывающим или кластерным), а во вторичном индексе сохраняются лишь поля, по которым построен вторичный индекс, и значения полей, участвующих в первичном индексе. Тогда при любом изменении значения, по которому построен вторичный ключ, приходится сначала удалять из вторичного индекса старый ключ, и только потом вставлять новый. Старое значение во время обновления неизвестно – именно его и нужно читать из первичного ключа с точки зрения внутреннего устройства.

Например:

```
update t1 set city='Moscow' where id=1
```

Чтобы уменьшить количество чтений с диска, особенно для несуществующих значений, практически все LSM-деревья используют вероятностные структуры данных. Tarantool не исключение. Классический фильтр Блума – это набор из нескольких (обычно 3-5) битовых массивов. При записи для каждого ключа вычисляется несколько хеш-функций, и в каждом массиве выставляется бит, соответствующий

значению хеша. При хешировании могут возникнуть коллизии, поэтому некоторые биты могут быть проставлены дважды. Интерес представляют биты, которые оказались не проставлены после записи всех ключей. При поиске также вычисляются выбранные хеш-функции. Если хотя бы в одном из битовых массивов бит не стоит, то значение в файле отсутствует. Вероятность срабатывания фильтра Блума определяется теоремой Байеса: каждая хеш-функция представляет собой независимую случайную величину, благодаря чему вероятность того, что во всех битовых массивах одновременно произойдет коллизия, очень мала.

Ключевым преимуществом реализации фильтров Блума в Tarantool является простота настройки. Единственный параметр, который можно менять независимо для каждого индекса, называется `vinyl_bloom_fpr` (FPR в данном случае означает сокращение от «false positive ratio» – коэффициент ложноположительного срабатывания), который по умолчанию равен 0,05, или 5%. На основе этого параметра Tarantool автоматически строит фильтры Блума оптимального размера для поиска как по полному ключу, так и по компонентам ключа. Сами фильтры Блума хранятся вместе с постраничным индексом в файле `.index` и кэшируются в оперативной памяти.

### Кэширование

Многие привыкли считать кэширование панацеей от всех проблем с производительностью: «В любой непонятной ситуации добавляй кэш». В vinyl'е мы смотрим на кэш скорее как на средство снижения общей нагрузки на диск, и, как следствие, получения более предсказуемого времени ответов на запросы, которые не попали в кэш. В vinyl'е реализован уникальный для транзакционных систем вид кэша под названием «кэш диапазона кортежей» (range tuple cache). В отличие от RocksDB, например, или MySQL, этот кэш хранит не страницы, а уже готовые диапазоны значений индекса, после их чтения с диска и слияния всех уровней. Это позволяет использовать кэш для запросов как по одному ключу, так и по диапазону ключей. Поскольку в кэше хранятся только горячие данные, а не, скажем, страницы (в странице может быть востребована лишь часть данных), оперативная память используется наиболее оптимально. Размер кэша задается в параметре `vinyl_cache`.

### Управление сборкой мусора

Возможно, добравшись до этого места вы уже начали терять концентрацию и нуждаетесь в заслуженной дозе допамина. Самое время сделать перерыв, так как для того, чтобы разобраться с оставшейся частью, понадобятся серьезные усилия.

В vinyl'е устройство одного LSM-дерева – это лишь фрагмент мозаики. Vinyl создает и обслуживает несколько LSM-деревьев даже для одной таблицы (так называемого спейса) – по одному дереву на каждый индекс. Но даже один единственный индекс может состоять из десятков LSM-деревьев. Попробуем разобраться, зачем.

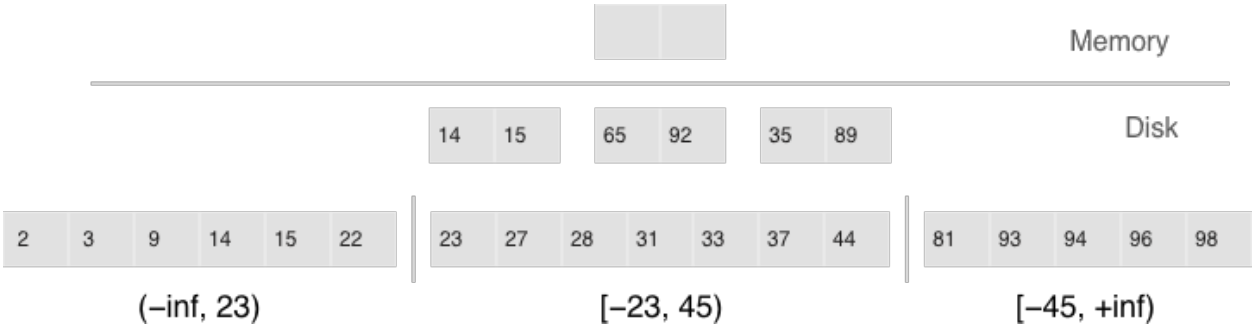
Рассмотрим наш стандартный пример: 100 000 000 записей по 100 байтов каждая. Через некоторое время на самом нижнем уровне LSM у нас может оказаться файл размером 10 ГБ. Во время слияния последнего уровня мы создадим временный файл, который также будет занимать около 10 ГБ. Данные на промежуточных уровнях тоже занимают место: по одному и тому же ключу дерево может хранить несколько операций. Суммарно для хранения 10 ГБ полезных данных нам может потребоваться до 30 ГБ свободного места: 10 ГБ на последний уровень, 10 ГБ на временный файл и 10 ГБ на всё остальное. А если данных не 1 ГБ, а 1 ТБ? Требовать, чтобы количество свободного места на диске всегда в несколько раз превышало объем полезных данных, экономически нецелесообразно, да и создание файла в 1ТБ может занимать десятки часов. При любой аварии или перезапуске системы операцию придется начинать заново.

Рассмотрим другую проблему. Представим, что первичный ключ дерева – это монотонная последовательность, например, временной ряд. В этом случае основные вставки будут приходиться на правую

часть диапазона ключей. Нет смысла заново производить слияние лишь для того, чтобы дописать в конец и без того огромного файла еще несколько миллионов записей.

А если вставки происходят, в основном, в одну часть диапазона ключей, а чтения – из другой части? Как в этом случае оптимизировать форму дерева? Если оно будет слишком высоким, пострадают чтения, если слишком низким – запись.

Tarantool «факторизует» проблему, создавая не одно, а множество LSM-деревьев для каждого индекса. Примерный размер каждого поддерева можно задать в конфигурационном параметре `vinyl_range_size`. Такие поддеревья называется диапазонами («range»).




Факторизация больших LSM-деревьев с помощью диапазонов

- Диапазоны отражают статичную структуру упорядоченных файлов
- Срезы объединяют упорядоченный файл в диапазон

Изначально, пока в индексе мало элементов, он состоит из одного диапазона. По мере добавления элементов суммарный объем может превысить *максимальный размер диапазона*. В таком случае выполняется операция под названием «разделение» (split), которая делит дерево на две равные части. Разделение происходит по срединному элементу диапазона ключей, хранящихся в дереве. Например, если изначально дерево хранит полный диапазон -inf... +inf, то после разделения по срединному ключу X получим два поддерева: одно будет хранить все ключи от -inf до X, другое – от X до +inf. Таким образом, при вставке или чтении мы однозначно знаем, к какому поддереву обращаться. Если в дереве были удаления и каждый из соседних диапазонов уменьшился, выполняется обратная операция под названием «объединение» (coalesce). Она объединяет два соседних дерева в одно.

Разделение и объединение не приводят к слиянию, созданию новых файлов и прочим тяжеловесным операциям. LSM-дерево – это лишь набор файлов. В vinyl'е мы реализовали специальный журнал метаданных, позволяющий легко отслеживать, какой файл принадлежит какому поддереву или поддеревьям. Журнал имеет расширение `.vylog`, по формату он совместим с файлом `.xlog`. Как и файл `.xlog`, происходит автоматическая ротация файла при каждой контрольной точке. Чтобы избежать повторного создания файлов при разделении и объединении, мы ввели промежуточную сущность – срез (slice). Это ссылка на файл с указанием диапазона значений ключа, которая хранится исключительно в журнале метаданных. Когда число ссылок на файл становится равным нулю, файл удаляется. А когда необходимо произвести разделение или объединение, Tarantool создает срезы для каждого нового дерева, старые срезы удаляет, и записывает эти операции в журнал метаданных. Буквально, журнал метаданных хранит записи вида <идентификатор дерева, идентификатор среза> или <идентификатор среза, идентификатор файла, мин, макс>.

Таким образом, непосредственно тяжелая работа по разбиению дерева на два поддерева, откладывается до слияния и выполняется автоматически. Огромным преимуществом подхода с разделением всего диапазона ключей на диапазоны является возможность независимо управлять размером L0, а также процессом создания дампов и слиянием для каждого поддерева. В результате эти процессы являются управляемыми и предсказуемыми. Наличие отдельного журнала метаданных также упрощает выпол-

нение таких операций, как усечение и удаление – в vinyl'е они обрабатываются мгновенно, потому что работают исключительно с журналом метаданных, а удаление мусора выполняется в фоне.

### Расширенные возможности vinyl'а

#### Upsert (обновление и вставка)

В предыдущих разделах упоминались лишь две операции, которые хранит LSM-дерево: удаление и замена. Давайте рассмотрим, как представлены все остальные. Вставку можно представить с помощью замены – необходимо лишь предварительно убедиться в отсутствии элемента указанным ключом. Для выполнения обновления необходимо предварительно считывать старое значение из дерева, так что и эту операцию проще записать в дерево как замену – это ускорит будущие чтения по этому ключу. Кроме того, обновление должно вернуть новое значение, так что скрытых чтений никак не избежать.

В B-деревьях скрытые чтения почти ничего не стоят: чтобы обновить блок, его в любом случае необходимо прочитать с диска. Для LSM-деревьев идея создания специальной операции обновления, которая не приводила бы к скрытым чтениям, выглядит очень заманчивой.

Такая операция должна содержать как значение по умолчанию, которое нужно вставить, если данных по ключу еще нет, так и список операций обновления, которые нужно выполнить, если значение существует.

На этапе выполнения транзакции Tarantool лишь сохраняет всю операцию в LSM-дереве, а «выполняет» ее уже только во время слияния.

Операция обновления и вставки:

```
space:upsert(tuple, {{operator, field, value}, ... })
```

- Обновление без чтения или вставка
- Отложенное выполнение
- Фоновое сжатие операций обновления и вставки предотвращает накапливание операций

К сожалению, если откладывать выполнение операции на этап слияния, возможностей для обработки ошибок не остается. Поэтому Tarantool стремится максимально проверять операции обновления и вставки upsert перед записью в дерево. Тем не менее, некоторые проверки можно выполнить лишь имея старые данные на руках. Например, если обновление прибавляет число к строке или удаляет несуществующее поле.

Операция с похожей семантикой присутствует во многих продуктах, в том числе в PostgreSQL и MongoDB. Но везде она представляет собой лишь синтаксический сахар, объединяющий обновление и вставку, не избавляя СУБД от необходимости выполнять скрытые чтения. Скорее всего, причиной этого является относительная новизна LSM-деревьев в качестве структур данных для хранения.

Хотя обновление и вставка upsert представляет собой очень важную оптимизацию, и ее реализация стоила нам долгой напряженной работы, следует признать, что ее применимость ограничена. Если в таблице есть вторичные ключи или триггеры, скрытых чтений не избежать. А если у вас есть сценарии, для которых не нужны вторичные ключи и обновление после завершения транзакции однозначно не приведет к ошибкам – эта операция для вас.

Небольшая история, связанная с этим оператором: vinyl только начинал «взрослеть», и мы впервые запустили операцию обновления и вставки upsert на рабочие серверы. Казалось бы, идеальные условия: огромный набор ключей, текущее время в качестве значения, операции обновления либо вставляют ключ, либо обновляют текущее время, редкие операции чтения. Нагрузочные тесты показали отличные результаты.

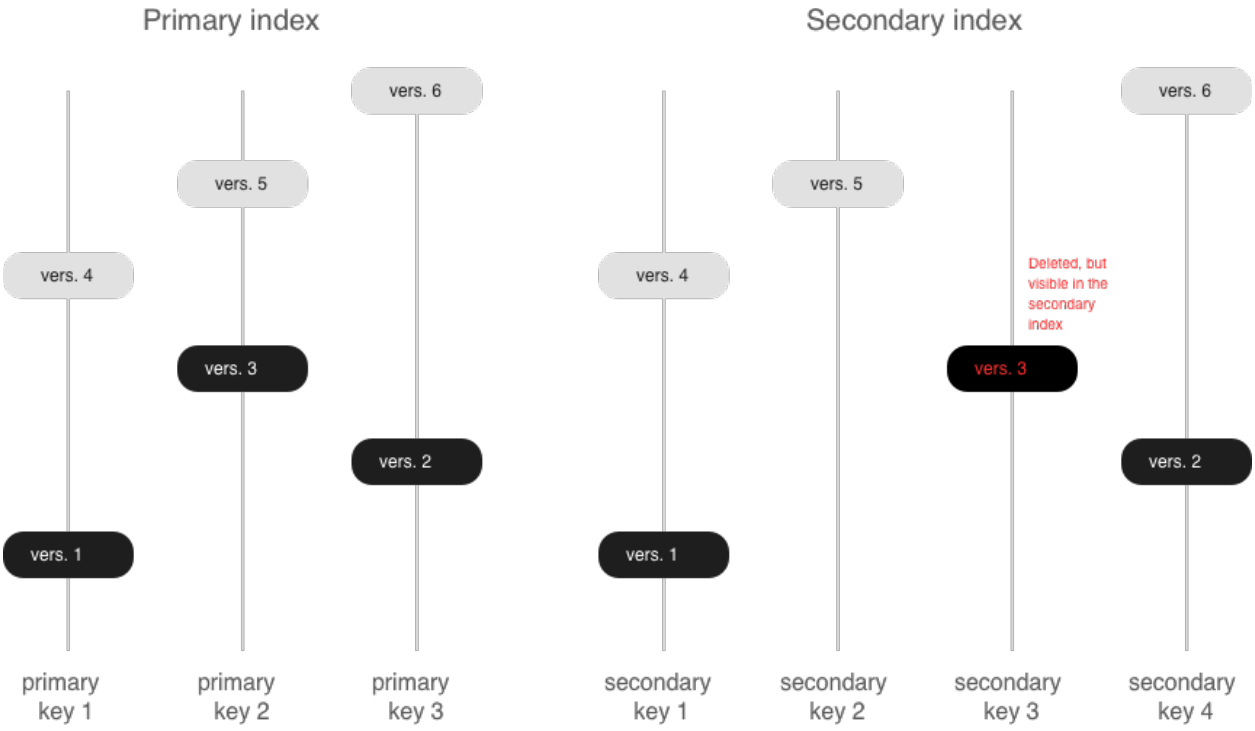
Тем не менее, после пары дней работы процесс Tarantool начал потреблять 100 % CPU, а производительность системы упала практически до нуля.

Начали подробно изучать проблему. Оказалось, что распределение запросов по ключам существенно отличалось от того, что мы видели в тестовом окружении. Оно было... очень неравномерное. Большая часть ключей обновлялась 1-2 раза за сутки, и база для них не была нагружена. Но были ключи гораздо более горячие – десятки тысяч обновлений в сутки. Tarantool прекрасно справлялся с этим потоком обновлений. А вот когда по ключу с десятком тысяч операций обновления и вставки upsert происходило чтение, всё шло под откос. Чтобы вернуть последнее значение, Tarantool приходилось каждый раз прочитать и «проиграть» историю из десятков тысяч команд обновления и вставки upsert. На стадии проекта мы надеялись, что это произойдет автоматически во время слияния уровней, но до слияния дело даже не доходило: памяти L0 было предостаточно, и дампы не создавались.

Решили мы проблему добавлением фонового процесса, осуществляющего упреждающие чтения для ключей, по которым накопилось больше нескольких десятков операций обновления и вставки upsert с последующей заменой на прочитанное значение.

### Вторичные ключи

Не только для операции обновления остро стоит проблема оптимизации скрытых чтений. Даже операция замены при наличии вторичных ключей вынуждена читать старое значение: его нужно независимо удалить из вторичных индексов, а вставка нового элемента может этого не сделать, оставив в индексе мусор.




Если вторичные индексы не уникальны, то удаление из них «мусора» также можно перенести в фазу слияния, что мы и делаем в Tarantool. Природа LSM-дерева, в котором файлы обновляются путем присоединения новых записей, позволила нам реализовать в vinyl'е полноценные сериализуемые транзакции. Запросы только для чтения при этом используют старые версии данных и не блокируют запись. Сам менеджер транзакций пока довольно простой: в традиционной классификации он реализует

класс MVTO (multiversion timestamp ordering – упорядочение временных меток на основе многоверсионности), при этом в конфликте побеждает та транзакция, что завершилась первой. Блокировок и свойственных им взаимоблокировок нет. Как ни странно, это скорее недостаток, чем преимущество: при параллельном выполнении можно повысить количество успешных транзакций, задерживая некоторые из них в нужный момент на блокировке. Развитие менеджера транзакций в наших ближайших планах. В текущей версии мы сфокусировались на том, чтобы сделать алгоритм корректным и предсказуемым на 100%. Например, наш менеджер транзакций – один из немногих в NoSQL-среде, поддерживающих так называемые «блокировки разрывов» (gap locks).

### Различие между движками memtx и vinyl

Основное различие между движками memtx и vinyl в том, что memtx — in-memory движок, тогда как vinyl — это дисковый движок. Обычно in-memory движок быстрее: время выполнения запроса, как правило, менее 1 мс. Поэтому движок memtx используется в Tarantool по умолчанию. Однако если база данных не помещается в доступную память, а дополнительную память добавить невозможно, то лучше использовать дисковый движок, в данном случае vinyl.

| Характеристика | memtx | vinyl |
|---|---|---|
| Поддерживаемый тип индекса | TREE, HASH, *RTREE* или BITSET | TREE |
| Временные спейсы | Поддерживается | Не поддерживается |
| функция *random()* | Поддерживается | Не поддерживается |
| функция *alter()* | Поддерживается | Поддерживается с версии 1.10.2 (первичный индекс изменять нельзя) |
| функция *len()* | Возвращает количество кортежей в спейсе | Возвращает максимальное примерное количество кортежей в спейсе |
| функция *count()* | Занимает одинаковые периоды времени | Занимает различное количество времени в зависимости от состояния БД |
| функция *delete()* | Возвращает удаленный кортеж, если есть таковой | Всегда возвращает nil |
| передача управления | Не передает управление на запросах выборки, если не происходит коммит транзакции в журнал упреждающей записи (WAL) | Передает управление на запросах выборки или аналогичных: get() или pairs() |

# ГЛАВА 5

## Вложенный модуль box.space

**CRUD operations** in Tarantool are implemented by the `box.space` submodule. It has the data-manipulation functions `select`, `insert`, `replace`, `update`, `upsert`, `delete`, `get`, `put`. It also has members, such as id, and whether or not a space is enabled.

Ниже приведен перечень всех функций и элементов модуля `box.space`.

| Имя | Использование |
|---|---|
| *box.schema.space.create()* | Создание спейса |
| *space_object:alter()* | Изменение спейса |
| *space_object:auto_increment()* | Генерация ключа + вставка кортежа |
| *space_object:bsize()* | Подсчет байтов |
| *space_object:count()* | Подсчет кортежей |
| *space_object:create_index()* | Создание индекса |
| *space_object:delete()* | Удаление кортежа |
| *space_object:drop()* | Удаление спейса |
| *space_object:format()* | Объявление имен и типов полей |
| *space_object:frommap()* | Конвертация ассоциативного массива в кортеж или таблицу |
| *space_object:get()* | Выбор кортежа |
| *space_object:insert()* | Вставка кортежа |
| *space_object:len()* | Подсчет кортежей |
| *space_object:on_replace()* | Создание триггера замены с функцией, которая не может изменять кортеж |
| *space_object:before_replace()* | Создание триггера замены с функцией, которая может изменять кортеж |
| *space_object:pairs()* | Подготовка к итерации |
| *space_object:put()* | Вставка или замена кортежа |
| *space_object:rename()* | Переименование спейса |
| *space_object:replace() / put()* | Вставка или замена кортежа |
| *space_object:run_triggers()* | Включение/отключение триггера замены |
| *space_object:select()* | Выбор одного или более кортежей |
| *space_object:truncate()* | Удаление всех кортежей |
| *space_object:update()* | Обновление кортежа |

continues on next page

Таблица 1 – продолжение с предыдущей страницы

| Имя | Использование |
|---|---|
| *space_object:upsert()* | Обновление кортежа |
| *space_object extensions* | Любая функция / метод, которые хочет добавить любой пользователь |
| *box.space.create_check_constraint()* | Создание проверочного ограничения |
| *space_object:enabled* | Флаг, если спейс активен – true |
| *space_object:field_count* | Необходимое количество полей |
| *space_object.id* | Числовой идентификатор спейса |
| *space_object.index* | Контейнер для индексов спейса |
| *box.space._cluster* | (Метаданные) Список наборов реплик |
| *box.space._func* | (Метаданные) Список кортежей с функциями |
| *box.space._index* | (Метаданные) Список индексов |
| *box.space._vindex* | (Метаданные) Список индексов, доступных текущему пользователю |
| *box.space._priv* | (Метаданные) Список прав |
| *box.space._vpriv* | (Метаданные) Список прав, доступных текущему пользователю |
| *box.space._schema* | (Метаданные) Список схем |
| *box.space._sequence* | (Метаданные) Список последовательностей |
| *box.space._sequence_data* | (Метаданные) Список последовательностей |
| *box.space._space* | (Метаданные) Список спейсов |
| *box.space._vspace* | (Метаданные) Список спейсов, доступных текущему пользователю |
| *box.space._space_sequence* | (Metadata) List of connections between spaces and sequences |
| *box.space._vspace_sequence* | (Metadata) List of connections between spaces and sequences accessible for the current user |
| *box.space._user* | (Метаданные) Список пользователей |
| *box.space._vuser* | (Метаданные) Список пользователей, доступных текущему пользователю |
| *box.space._ck_constraint* | (Метаданные) Список проверочных ограничений |
| *box.space._collation* | (Метаданные) Список видов сортировки |
| *box.space._vcollation* | (Метаданные) Список видов сортировки, доступных текущему пользователю |
| *Представления системных спейсов* | (Метаданные) Спейсы, имена которых начинаются с _v |
| *box.space._session_settings* | (Метаданные) Список настроек, которые влияют на поведение текущего сеанса |

См. также *практические примеры работы с CRUD-операциями*.

## 5.1 box.schema.space.create()

`box.schema.space.create(`*space-name*[, *{space_opts}*]`)`
`box.schema.create_space(`*space-name*[, *{space_opts}*]`)`

Create a *space*. You can use either syntax. For example, `s = box.schema.space.create('tester')` has the same effect as `s = box.schema.create_space('tester')`.

There are *three syntax variations* for object references targeting space objects, for example `box.schema.space.drop(`*space-id*`)` drops a space. However, the common approach is to use functions attached to the space objects, for example *space_object:drop()*.

Следующим шагом после создания спейса будет *создание индекса* для него, после чего можно будет выполнять вставку, выборку и другие функции *box.space*.

**Параметры**

- `space-name` (*`string`*) – имя спейса, которое должно соответствовать *правилам именования объектов*
- `options` (*`table`*) – space options (see *space_opts*)

**возвращает** объект спейса

**тип возвращаемого значения** пользовательские данные

### 5.1.1 space_opts

`object space_opts`

Space options that include the space id, format, field count, constraints and foreign keys, and so on. These options are passed to the *box.schema.space.create()* method.

---

**Примечание:** These options are also passed to *space_object:alter()*.

---

`space_opts.if_not_exists`

Create a space only if a space with the same name does not exist already. Otherwise, do nothing but do not cause an error.

Type: boolean

Default: `false`

`space_opts.engine`

A *storage engine*.

Type: string

Default: `memtx`

Possible values: `memtx`, `vinyl`

`space_opts.id`

A unique numeric identifier of the space: users can refer to spaces with this id instead of the name.

Type: number

Default: last space's ID + 1

`space_opts.field_count`

A fixed count of *fields*. For example, if `field_count=5`, it is illegal to insert a tuple with fewer than or more than 5 fields.

Type: number

Default: `0` (not fixed)

`space_opts.user`
The name of the user who is considered to be the space's *owner* for authorization purposes.

Type: string
Default: current user's name

`space_opts.format`
Field names and types. See the illustrations of format clauses in the *space_object:format()* description and in the *box.space._space* example. Optional and usually not specified.

Type: table
Default: blank

`space_opts.is_local`
Space contents are *replication-local*: changes are stored in the *write-ahead log* of the local node but there is no *replication*.

Type: boolean
Default: `false`

`space_opts.temporary`
Space contents are temporary: changes are not stored in the *write-ahead log* and there is no *replication*.

---

**Примечание:** Vinyl does not support temporary spaces.

---

Type: boolean
Default: `false`

`space_opts.is_sync`
Any transaction doing a DML request on this space becomes *synchronous*.

**Пример:**

Type: boolean
Default: `false`

`space_opts.constraint`
The *constraints* that space tuples must satisfy.

Type: table

Default: blank

**Пример:**

```
-- Define a tuple constraint function --
box.schema.func.create('check_person', {
    language = 'LUA',
    is_deterministic = true,
    body = 'function(t, c) return (t.age >= 0 and #(t.name) > 3) end'
})

-- Create a space with a tuple constraint --
customers = box.schema.space.create('customers', {constraint = 'check_person'})
```

space_opts.foreign_key
The *foreign keys* for space fields.

Type: table
Default: blank

**Пример:**

```
-- Create a space with a tuple foreign key --
box.schema.space.create("orders", {
    foreign_key = {
        space = 'customers',
        field = {customer_id = 'id', customer_name = 'name'}
    }
})

box.space.orders:format({
    {name = "id", type = "number"},
    {name = "customer_id" },
    {name = "customer_name"},
    {name = "price_total", type = "number"},
})
```

Если выполнить `box.cfg{read_only=true...}` во время *конфигурации* по-разному влияет на спейсы в зависимости от опций, использованных во время `box.schema.space.create`, как описано в таблице:

| Характеристика | Можно создать? | Допускает запись? | Реплицируется? | Сохраняется? |
|---|---|---|---|---|
| (по умолчанию) | нет | нет | да | да |
| temporary (временный) | нет | да | нет | нет |
| is_local | нет | да | нет | да |

**Пример:**

```
tarantool> s = box.schema.space.create('space55')
---
...
tarantool> s = box.schema.space.create('space55', {
         >   id = 555,
         >   temporary = false
         > })
---
- error: Space 'space55' already exists
...
tarantool> s = box.schema.space.create('space55', {
         >   if_not_exists = true
         > })
---
...
```

## 5.2 space_object:alter()

object space_object

space_object:alter(*options*)

Для версии *2.5.2* и выше. Изменение существующего спейса. Этот метод меняет определенные параметры спейса.

**Параметры**

- options (*table*) – the space options such as field_count, user, format, name, and other. The full list of these options with descriptions parameters is provided in *box.schema.space.create()*

**возвращается** ничего в случае успеха; ошибка при неудаче

**Пример:**

```
tarantool> s = box.schema.create_space('tester')
---
...
tarantool> format = {{name = 'field1', type = 'unsigned'}}
---
...
tarantool> s:alter({name = 'tester1', format = format})
---
...
tarantool> s.name
---
- tester1
...
tarantool> s:format()
---
- [{'name': 'field1', 'type': 'unsigned'}]
...
```

## 5.3 space_object:auto_increment()

object space_object

space_object:auto_increment(*tuple*)

Вставка нового кортежа, используя первичный ключ с автоматическим увеличением. В спейсе, указанном через space_object должен быть первичный TREE-индекс типа *„unsigned“ или „integer“, или „number“*. Поле первичного ключа будет увеличиваться перед вставкой.

Данный метод объявлен устаревшим с версии 1.7.5 – лучше использовать *последовательности*.

**Параметры**

- space_object (*space_object*) – *ссылка на объект*
- tuple (table/tuple) – поля кортежа, не включая поле первичного ключа

**возвращает** вставленный кортеж.

**тип возвращаемого значения** кортеж

**Факторы сложности** Размер индекса, тип индекса, количество индексов, к которым получен доступ, *настройки журнала упреждающей записи (WAL)*.

**Возможные ошибки:**

- неподходящий тип индекса;
- проиндексированное поле первичного ключа не является числовым.

**Пример:**

```
tarantool> box.space.tester:auto_increment{'Fld#1', 'Fld#2'}
---
- [1, 'Fld#1', 'Fld#2']
...
tarantool> box.space.tester:auto_increment{'Fld#3'}
---
- [2, 'Fld#3']
...
```

## 5.4 space_object:bsize()

object space_object

space_object:bsize()

**Параметры**

- space_object (*space_object*) – *ссылка на объект*

**возвращает** Количество байтов в спейсе. Это число, которое хранится во внутренней памяти Tarantool, представляет собой общее количество байтов во всех кортежах, не включая ключи индекса. Для получения информации об измерении размера индекса, см. *index_object:bsize()*.

**Пример:**

```
tarantool> box.space.tester:bsize()
---
- 22
...
```

## 5.5 space_object:count()

object space_object

space_object:count(*key* [, *iterator* ])

Возврат количества кортежей. Если сравнивать с *len()*, то count() работает медленнее, поскольку сканирует весь спейс для подсчета кортежей.

**Параметры**

- space_object (*space_object*) – *ссылка на объект*
- key (scalar/table) – значения поля первичного ключа, которые должны возвращаться в виде Lua-таблицы, если ключ составной
- iterator – метод сопоставления

**возвращает** Количество кортежей.

**Возможные ошибки:**

- ER_TRANSACTION_CONFLICT, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Пример:**

```
tarantool> box.space.tester:count(2, {iterator='GE'})
---
- 1
...
```

## 5.6 space_object:create_index()

object space_object

space_object:create_index(*index-name* [, *index_opts* ])

Создание *индекса*.

Обязательным условием является создание индекса для спейса до вставки в спейс кортежей или выборки кортежей из него. Первый созданный индекс, который будет использоваться в качестве первичного индекса, должен быть уникальным.

**Параметры**

- space_object (*space_object*) – *ссылка на объект*
- index_name (*string*) – имя индекса, которое должно соответствовать *правилам именования объектов*

- index_opts (*table*) – index options (see *index_opts*)

**возвращает** объект индекса

**тип возвращаемого значения** *index_object*

**Возможные ошибки:**

- too many parts
- index „. . .“ already exists
- primary key must be unique

Building or rebuilding a large index will cause occasional *yields* so that other requests will not be blocked. If the other requests cause an illegal situation such as a duplicate key in a unique index, building or rebuilding such index will fail.

**Пример:**

```
-- Create a space --
bands = box.schema.space.create('bands')

-- Specify field names and types --
box.space.bands:format({
    { name = 'id', type = 'unsigned' },
    { name = 'band_name', type = 'string' },
    { name = 'year', type = 'unsigned' }
})

-- Create a primary index --
box.space.bands:create_index('primary', { parts = { 'id' } })

-- Create a unique secondary index --
box.space.bands:create_index('band', { parts = { 'band_name' } })

-- Create a non-unique secondary index --
box.space.bands:create_index('year', { parts = { { 'year' } }, unique = false })

-- Create a multi-part index --
box.space.bands:create_index('year_band', { parts = { { 'year' }, { 'band_name'␣
↪} } })
```

### 5.6.1 index_opts

object index_opts

Index options that include the index name, type, identifiers of key fields, and so on. These options are passed to the *space_object.create_index()* method.

---

**Примечание:** These options are also passed to *index_object:alter()*.

---

index_opts.type

The *index type*.

Type: string

Default: `TREE`
Possible values: `TREE`, `HASH`, `RTREE`, `BITSET`

`index_opts.id`
A unique numeric identifier of the index, which is generated automatically.

Type: number
Default: last index's ID + 1

`index_opts.unique`
Specify whether an index may be unique. When `true`, the index cannot contain the same key value twice.

Type: boolean
Default: `true`

**Пример:**

```
-- Create a non-unique secondary index --
box.space.bands:create_index('year', { parts = { { 'year' } }, unique = false })
```

`index_opts.if_not_exists`
Specify whether to swallow an error on an attempt to create an index with a duplicated name.

Type: boolean
Default: `false`

`index_opts.parts`
Specify the index's key parts.

Type: a table of *key_part* values
Default: `{1, 'unsigned'}`

**Пример:**

```
-- Create a primary index --
box.space.bands:create_index('primary', { parts = { 'id' } })

-- Create a unique secondary index --
box.space.bands:create_index('band', { parts = { 'band_name' } })

-- Create a non-unique secondary index --
box.space.bands:create_index('year', { parts = { { 'year' } }, unique = false })
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Create a multi-part index --
box.space.bands:create_index('year_band', { parts = { { 'year' }, { 'band_name'␣
↪} } })
```

---

**Примечание: Другой способ объявления оператора parts**

Если раньше (до версии *2.7.1*) индекс состоял из одной части, содержал дополнительные параметры, например `is_nullable` или `collation`, и был описан так:

```
my_space:create_index('one_part_idx', {parts = {1, 'unsigned', is_nullable=true}
↪})
```

(с одинарными фигурными скобками), то Tarantool игнорировал эти параметры.

Начиная с версии *2.7.1*, при описании индекса можно не указывать дополнительные фигурные скобки, но допускаются оба варианта:

```
-- с дополнительными фигурными скобками
my_space:create_index('one_part_idx', {parts = {{1, 'unsigned', is_
↪nullable=true}}})

-- без дополнительных фигурных скобок
my_space:create_index('one_part_idx', {parts = {1, 'unsigned', is_nullable=true}
↪})
```

---

`index_opts.dimension`
The RTREE index dimension.

Type: number
Default: 2

`index_opts.distance`
The RTREE index distance type.

Type: string
Default: `euclid`
Possible values: `euclid`, `manhattan`

`index_opts.sequence`
Create a generator for indexes using a sequence object. Learn more from *specifying a sequence in create_index()*.

Type: string or number

`index_opts.func`
Specify the identifier of the *functional index* function.

Type: string

`index_opts.hint`
**Since:** *2.6.1*

Specify whether hint optimization is enabled for the TREE index:

- If `true`, the index works faster.
- If `false`, the index size is reduced by half.

Type: boolean
Default: `true`

`index_opts.bloom_fpr`
**Vinyl only**

Specify the bloom filter's false positive rate.

Type: number
Default: *vinyl_bloom_fpr*

`index_opts.page_size`
**Vinyl only**

Specify the size of a page used for read and write disk operations.

Type: number
Default: *vinyl_page_size*

`index_opts.range_size`
**Vinyl only**

Specify the default maximum range size (in bytes) for a vinyl index.

Type: number
Default: *vinyl_range_size*

`index_opts.run_count_per_level`
**Vinyl only**

Specify the maximum number of runs per level in the LSM tree.

Type: number
Default: *vinyl_run_count_per_level*

`index_opts.run_size_ratio`
**Vinyl only**

Specify the ratio between the sizes of different levels in the LSM tree.

Type: number
Default: *vinyl_run_size_ratio*

### 5.6.2 key_part

`object key_part`
A descriptor of a single part in a multipart key. A table of parts is passed to the *index_opts.parts* option.

`key_part.field`
Specify the field number or name.

---

**Примечание:** To create a key part by a field name, you need to specify *space_object:format()* first.

---

Type: string or number

**Examples:** *Creating an index using field names and numbers*

`key_part.type`
Specify the field type. If the field type is specified in *space_object:format()*, `key_part.type` inherits this value.

Type: string
Default: `scalar`
Possible values: listed in *Indexed field types*

`key_part.collation`
Specify the *collation* used to compare field values. If the field collation is specified in *space_object:format()*, `key_part.collation` inherits this value.

Type: string
Possible values: listed in the *box.space._collation* system space

**Пример:**

```
-- Create a space --
box.schema.space.create('tester')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Use the 'unicode' collation --
box.space.tester:create_index('unicode', { parts = { { field = 1,
                                                        type = 'string',
                                                        collation = 'unicode' }␣
↪} })

-- Use the 'unicode_ci' collation --
box.space.tester:create_index('unicode_ci', { parts = { { field = 1,
                                                        type = 'string',
                                                        collation = 'unicode_ci
↪' } } })

-- Insert test data --
box.space.tester:insert { 'ЕЛЕ' }
box.space.tester:insert { 'елейный' }
box.space.tester:insert { 'ёлка' }

-- Returns nil --
select_unicode = box.space.tester.index.unicode:select({ 'ЁлКа' })
-- Returns 'ёлка' --
select_unicode_ci = box.space.tester.index.unicode_ci:select({ 'ЁлКа' })
```

`key_part.is_nullable`

Specify whether `nil` (or its equivalent such as `msgpack.NULL`) can be used as a field value. If the `is_nullable` option is specified in *space_object:format()*, `key_part.is_nullable` inherits this value.

You can set this option to `true` if:

- the index type is TREE
- the index is not the primary index

It is also legal to insert nothing at all when using trailing nullable fields. Within indexes, such null values are always treated as equal to other null values and are always treated as less than non-null values. Nulls may appear multiple times even in a unique index.

Type: boolean
Default: false

**Пример:**

```
box.space.tester:create_index('I', {unique = true, parts = {{field = 2, type =
↪'number', is_nullable = true}}})
```

**Предупреждение:** It is legal to create multiple indexes for the same field with different `is_nullable` values or to call *space_object:format()* with a different `is_nullable` value from what is used for an index. When there is a contradiction, the rule is: null is illegal unless `is_nullable=true` for every index and for the space format.

`key_part.exclude_null`
**Since:** *2.8.2*

Specify whether an index can skip tuples with null at this key part. You can set this option to `true` if:

- the index type is TREE
- the index is not the primary index

If `exclude_null` is set to `true`, `is_nullable` is set to `true` automatically. Note that this option can be changed dynamically. In this case, the index is rebuilt.

Такие индексы вообще не хранят отфильтрованные кортежи, поэтому индексирование будет выполняться быстрее.

Type: boolean
Default: false

`key_part.path`
Specify the path string for a map field.

Type: string

See the examples below:

- *Creating an index using the path option for map fields*
- *Creating a multikey index using the path option with [*]*

### 5.6.3 Примеры

#### Creating an index using field names and numbers

`create_index()` can use field names or field numbers to define key parts.

**Example 1 (field names):**

To create a key part by a field name, you need to specify *space_object:format()* first.

```
-- Create a primary index --
box.space.bands:create_index('primary', { parts = { 'id' } })

-- Create a unique secondary index --
box.space.bands:create_index('band', { parts = { 'band_name' } })

-- Create a non-unique secondary index --
box.space.bands:create_index('year', { parts = { { 'year' } }, unique = false })

-- Create a multi-part index --
box.space.bands:create_index('year_band', { parts = { { 'year' }, { 'band_name' } } })
```

**Example 2 (field numbers):**

```
-- Create a primary index --
box.space.bands:create_index('primary', { parts = { 1 } })

-- Create a unique secondary index --
box.space.bands:create_index('band', { parts = { 2 } })

-- Create a non-unique secondary index --
box.space.bands:create_index('year', { parts = { { 3 } }, unique = false })

-- Create a multi-part index --
box.space.bands:create_index('year_band', { parts = { 3, 2 } })
```

### Создание индекса с использованием пути для полей с ассоциативными массивами (индексы по пути JSON)

Чтобы создать индекс для поля, которое представляет собой ассоциативный массив (строка с путем и скалярное значение), укажите строку с путем во время создания индекса:

`parts = {field-number, 'data-type', path = 'path-name'}`

Тип индекса должен быть TREE или HASH, а содержимое поля — всегда ассоциативный массив с одним и тем же путем.

**Пример 1 — Простое использование пути:**

```
box.schema.space.create('space1')
box.space.space1:create_index('primary', { parts = { { field = 1,
                                                        type = 'scalar',
                                                        path = 'age' } } })
box.space.space1:insert({ { age = 44 } })
box.space.space1:select(44)
```

**Пример 2 — для большей наглядности используем path вместе с format() и JSON-синтаксисом:**

```
box.schema.space.create('space2')
box.space.space2:format({ { 'id', 'unsigned' }, { 'data', 'map' } })
box.space.space2:create_index('info', { parts = { { 'data.full_name["firstname"]', 'str'␣
↪},
                                                  { 'data.full_name["surname"]', 'str' }␣
↪} })
box.space.space2:insert({ 1, { full_name = { firstname = 'John', surname = 'Doe' } } })
box.space.space2:select { 'John' }
```

### Создание индекса по массивам (multikey) с использованием опции path с символом [*]

Строка в параметре пути может содержать символ `[*]`, который называется **заменителем индекса массива**. Описанные так индексы используются для JSON-документов, у которых одинаковая структура.

Например, при создании индекса по полю №2 для документа со строками, который будет начинаться с `{'data': [{'name': '...'}, {'name': '...'}]`, раздел parts в запросе на создание индекса будет выглядеть так:

```
parts = {{field = 2, type = 'str', path = 'data[*].name'}}
```

Тогда кортежи с именами можно быстро получить с помощью `index_object:select({key-value})`.

A single field can have multiple keys, as in this example which retrieves the same tuple twice because there are two keys „A“ and „B“ which both match the request:

```
my_space = box.schema.space.create('json_documents')
my_space:create_index('primary')
multikey_index = my_space:create_index('multikey', {parts = {{field = 2, type = 'str',␣
↪path = 'data[*].name'}}})
my_space:insert({1,
         {data = {{name = 'A'},
                  {name = 'B'}},
          extra_field = 1}})
multikey_index:select({''}, {iterator = 'GE'})
```

Результат выборки будет выглядеть так:

```
tarantool> multikey_index:select({''},{iterator='GE'})
---
- - [1, {'data': [{'name': 'A'}, {'name': 'B'}], 'extra_field': 1}]
- [1, {'data': [{'name': 'A'}, {'name': 'B'}], 'extra_field': 1}]
...
```

The following restrictions exist:

- символ `[*]` должен стоять отдельно или в конце имени в пути.
- символ `[*]` не должен повторяться в пути.
- Если в индексе есть путь с `x[*]`, то в никаком другом индексе не может быть пути с x.компонентом,
- `[*]` нельзя указывать в пути первичного ключа.
- Если индекс должен быть уникальным (`unique=true`) и в нем есть путь с символом `[*]`, то запрещается использовать дублирующиеся ключи в разных кортежах, но в в одном кортеже можно использовать дублирующиеся ключи.
- Структура значения поля должна соответствовать стуктуре, заданной в определении пути, или значение поля должно быть nil (nil не индексируется).
- В спейсе с индексами по массивам можно хранить не более *~8000 элементов*, проиндексированных таким образом.

### Создание функционального индекса

Функциональные индексы — это индексы, которые вызывают пользовательскую функцию для формирования ключа индекса, в отличие от других типов индексов, где ключ формирует сам Tarantool. Функциональные индексы используют для сжатия, усечения или реверсирования или любого другого изменения индекса по желанию пользователя.

Ниже приведены рекомендации по созданию функциональных индексов:

- The function definition must expect a tuple, which has the contents of fields at the time a data-change request happens, and must return a tuple, which has the contents that will be put in the index.
- The `create_index` definition must include the specification of all key parts, and the custom function must return a table that has the same number of key parts with the same types.
- Спейс должен быть на движке memtx.
- Функция должна быть персистентной и детерминированной (см. *Создание функции с телом*).
- Части ключа не должны зависеть от JSON-путей.
- Функция должна получать доступ к значениям частей ключа по индексу, а не по имени поля.
- Функциональные индексы не должны быть по первичному ключу.
- Нельзя изменить ни функциональные индексы, ни функцию, если она используется для индекса, то есть единственный способ изменить их — это удалить индекс и создать его заново.
- Только функции, *запущенные из песочницы*, могут использоваться в функциональных индексах.

**Пример:**

Функция может создать ключ, используя только первую букву строкового поля.

1. Создайте спейс. В спейсе должно быть поле с первичным ключом, которое не будет полем для функционального индекса:

```
box.schema.space.create('tester')
box.space.tester:create_index('i', { parts = { { field = 1, type = 'string' } } })
```

2. Создайте функцию. Функция принимает кортеж. В этом примере она работает на кортеже tuple[2], поскольку источник ключа — поле номер 2, в которое мы вставляем данные. Используйте `string.sub()` из модуля `string`, чтобы получить первый символ:

```
function_code = [[function(tuple) return {string.sub(tuple[2],1,1)} end]]
```

3. Сделайте функцию персистентной с помощью `box.schema.create`:

```
box.schema.func.create('my_func',
        { body = function_code, is_deterministic = true, is_sandboxed = true })
```

4. Create a functional index. Specify the fields whose values will be passed to the function. Specify the function:

```
box.space.tester:create_index('func_index', { parts = { { field = 1, type = 'string
↪' } },
                                              func = 'my_func' })
```

5. Insert a few tuples. Select using only the first letter, it will work because that is the key. Or, select using the same function as was used for insertion:

```
box.space.tester:insert({ 'a', 'wombat' })
box.space.tester:insert({ 'b', 'rabbit' })
box.space.tester.index.func_index:select('w')
box.space.tester.index.func_index:select(box.func.my_func:call({ { 'tester', 'wombat
↪' } }))
```

Результаты двух запросов `select` будут выглядеть так:

```
tarantool> box.space.tester.index.func_index:select('w')
---
- - ['a', 'wombat']
...
tarantool> box.space.tester.index.func_index:select(box.func.my_func:call({{'tester
↪','wombat'}}));
---
- - ['a', 'wombat']
...
```

Вот пример кода полностью:

```
box.schema.space.create('tester')
box.space.tester:create_index('i', { parts = { { field = 1, type = 'string' } }
↪ })
function_code = [[function(tuple) return {string.sub(tuple[2],1,1)} end]]
box.schema.func.create('my_func',
        { body = function_code, is_deterministic = true, is_sandboxed = true })
box.space.tester:create_index('func_index', { parts = { { field = 1, type =
↪'string' } },
                                              func = 'my_func' })
box.space.tester:insert({ 'a', 'wombat' })
box.space.tester:insert({ 'b', 'rabbit' })
box.space.tester.index.func_index:select('w')
box.space.tester.index.func_index:select(box.func.my_func:call({ { 'tester',
↪'wombat' } }))
```

Функции для функциональных индексов могут возвращать **множество ключей**. Такие функции называют «мультиключевыми» (multikey).

To create a multikey function, the options of `box.schema.func.create()` must include `is_multikey = true`. The return value must be a table of tuples. If a multikey function returns N tuples, then N keys will be added to the index.

**Пример:**

```
tester = box.schema.space.create('withdata')
tester:format({ { name = 'name', type = 'string' },
                { name = 'address', type = 'string' } })
name_index = tester:create_index('name', { parts = { { field = 1, type =
↪'string' } } })
function_code = [[function(tuple)
       local address = string.split(tuple[2])
       local ret = {}
       for _, v in pairs(address) do
         table.insert(ret, {utf8.upper(v)})
```

(continues on next page)

(продолжение с предыдущей страницы)

```
       end
       return ret
     end]]
box.schema.func.create('address',
        { body = function_code,
          is_deterministic = true,
          is_sandboxed = true,
          is_multikey = true })
addr_index = tester:create_index('addr', { unique = false,
                                           func = 'address',
                                           parts = { { field = 1, type =
↪'string',
                                                     collation = 'unicode_ci' } }␣
↪})
tester:insert({ "James", "SIS Building Lambeth London UK" })
tester:insert({ "Sherlock", "221B Baker St Marylebone London NW1 6XE UK" })
addr_index:select('Uk')
```

## 5.7 space_object:delete()

object space_object

space_object:delete(*key*)

Delete a tuple identified by the primary key.

**Параметры**

- space_object (*space_object*) – *ссылка на объект*
- key (scalar/table) – значения поля первичного ключа, которые должны возвращаться в виде Lua-таблицы, если ключ составной

**возвращает** удаленный кортеж.

**тип возвращаемого значения** tuple

**Возможные ошибки:**

- ER_TRANSACTION_CONFLICT, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Факторы сложности:** Размер индекса, тип индекса

**Note regarding storage engine:** vinyl will return nil, rather than the deleted tuple.

**Пример:**

```
tarantool> box.space.tester:delete(1)
---
- [1, 'My first tuple']
...
tarantool> box.space.tester:delete(1)
---
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> box.space.tester:delete('a')
---
- error: 'Supplied key type of part 0 does not match index part type:
  expected unsigned'
...
```

Для получения дополнительной информации о сценариях использования и типичных ошибках, см. *Пример: использование операций с данными* далее в разделе.

## 5.8 space_object:drop()

`object space_object`

`space_object:drop()`

Удаление спейса. Метод реализуется в фоновом режиме и не блокирует последующие запросы.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*

**возвращает** nil

**Возможные ошибки:** `space_object` не существует.

**Факторы сложности** Размер индекса, тип индекса, количество индексов, к которым получен доступ, настройки журнала упреждающей записи (WAL).

**Пример:**

```
box.space.space_that_does_not_exist:drop()
```

## 5.9 space_object:format()

`object space_object`

`space_object:format([format-clause])`

Объявление имен и *типов* полей.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*
- `format-clause` (*table*) – список имен и типов полей

**возвращает** `nil`, если не указан оператор формата

**Возможные ошибки:**

- `space_object` does not exist
- field names are duplicated
- type is not legal

**Примечание:** Если вам нужно выполнить миграцию схемы, обратитесь к разделу *Миграции*.

Как правило, Tarantool допускает поля без имен и без указания типа. Но с помощью `format` можно, например, задокументировать, что N-ное поле представляет собой поле для фамилии и должно содержать строковое значение. Также оператор формата можно указать в *box.schema.space.create()*.

Оператор формата для каждого поля содержит определение в фигурных скобках: `{name='...',type='...'[,is_nullable=...]}`, где:

- The `name` value may be any string, provided that two fields do not have the same name.
- The `type` value may be any of allowed types: any | unsigned | string | integer | number | varbinary | boolean | double | decimal | uuid | array | map | scalar, but for creating an index use only *indexed fields*;
- (Optional) The `is_nullable` boolean value specifies whether `nil` can be used as a field value. See also: *key_part.is_nullable*.
- (Optional) The `collation` string value specifies the *collation* used to compare field values. See also: *key_part.collation*.
- (Optional) The `constraint` table specifies the *constraints* that the field value must satisfy.
- (Optional) The `foreign_key` table specifies the *foreign keys* for the field.

В кортежах недопустимы значения неправильного типа. Пример ниже вызовет ошибку:

```
--Этот пример вызовет ошибку.
box.space.tester:format({{' ',type='number'}})
box.space.tester:insert{'string-which-is-not-a-number'}
```

В кортежах недопустимы нулевые значения, если `is_nullable=false`, что задано по умолчанию. Пример ниже вызовет ошибку:

```
--Этот пример вызовет ошибку.
box.space.tester:format({{' ',type='number',is_nullable=false}})
box.space.tester:insert{nil,2}
```

В кортежах может быть больше полей, чем описано в операторе формата. Чтобы ограничить количество полей, необходимо указать для спейса элемент *field_count*.

В кортежах может быть меньше полей, чем описано в операторе формата, если пропущенные завершающие поля описаны с помощью `is_nullable=true`. Например, запрос ниже не приведет к ошибке формата:

```
box.space.tester:format({{'a',type='number'},{'b',type='number',is_
↪nullable=true}})
box.space.tester:insert{2}
```

Можно использовать `format` для спейса, в котором уже определен формат, заменяя таким образом предыдущие определения при условии, что нет конфликта с существующими данными или определениями индекса.

Можно использовать `format` для того, чтобы изменить значение флага `is_nullable`. Пример ниже не вызовет ошибки – и не приведет к перестроению спейса.

```
box.space.tester:format({{' ',type='scalar',is_nullable=false}})
box.space.tester:format({{' ',type='scalar',is_nullable=true}})
```

Но обратное изменение значение `is_nullable` с `true` на `false` может вызвать перестроение и привести к ошибке, если уже есть кортежи с нулевыми значениями.

**Пример:**

```
box.space.tester:format({{name='surname',type='string'},{name='IDX',type='array
↪'}})
box.space.tester:format({{name='surname',type='string',is_nullable=true}})
```

Можно использовать следующие варианты оператора:

- пропуск и „name=“, и „type=“,
- пропуск только „type=“,
- добавление дополнительных фигурных скобок.

В следующем примере иллюстрируются все варианты, первый для поля с именем „x“, второй – для двух полей с именами „x“ и „y“.

```
box.space.tester:format({{name='x',type='scalar'}})
box.space.tester:format({{name='x',type='scalar'},{name='y',type='unsigned'}})

box.space.tester:format({{'x'}})
box.space.tester:format({{'x'},{'y'}})

-- типы
box.space.tester:format({{name='x'}})
box.space.tester:format({{name='x'},{name='y'}})

box.space.tester:format({{'x',type='scalar'}})
box.space.tester:format({{'x',type='scalar'},{'y',type='unsigned'}})

box.space.tester:format({{'x','scalar'}})
box.space.tester:format({{'x','scalar'},{'y','unsigned'}})
```

В следующем примере показывается создание спейса, определение формата для него со всеми возможными типа и вставка данных.

```
tarantool> box.schema.space.create('t')
---
- engine: memtx
  before_replace: 'function: 0x4019c488'
  on_replace: 'function: 0x4019c460'
  ck_constraint: []
  field_count: 0
  temporary: false
  index: []
  is_local: false
  enabled: false
  name: t
  id: 534
- created
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> ffi = require('ffi')
---
...
tarantool> decimal = require('decimal')
---
...
tarantool> uuid = require('uuid')
---
...
tarantool> box.space.t:format({{name = '1', type = 'any'},
         >                     {name = '2', type = 'unsigned'},
         >                     {name = '3', type = 'string'},
         >                     {name = '4', type = 'number'},
         >                     {name = '5', type = 'double'},
         >                     {name = '6', type = 'integer'},
         >                     {name = '7', type = 'boolean'},
         >                     {name = '8', type = 'decimal'},
         >                     {name = '9', type = 'uuid'},
         >                     {name = 'a', type = 'scalar'},
         >                     {name = 'b', type = 'array'},
         >                     {name = 'c', type = 'map'}})
---
...
tarantool> box.space.t:create_index('i',{parts={2, type = 'unsigned'}})
---
- unique: true
  parts:
  - type: unsigned
    is_nullable: false
    fieldno: 2
  id: 0
  space_id: 534
  type: TREE
  name: i
...
tarantool> box.space.t:insert{{'a'}, -- any
         >                    1, -- unsigned
         >                    'W?', -- string
         >                    5.5, -- number
         >                    ffi.cast('double', 1), -- double
         >                    -0, -- integer
         >                    true, -- boolean
         >                    decimal.new(1.2), -- decimal
         >                    uuid.new(), -- uuid
         >                    true, -- scalar
         >                    {{'a'}}, -- array
         >                    {val=1}} -- map
---
- [['a'], 1, 'W?', 5.5, 1, 0, true, 1.2, 1f41e7b8-3191-483d-b46e-1aa6a4b14557,␣
↪true, [['a']], {'val': 1}]
...
```

Имена, указанные с помощью оператора формата, можно использовать в *space_object:get()*, в *space_object:create_index()*, в *tuple_object[field-name]* и в *tuple_object[field-path]*.

Если оператор формата не указан, то вернется таблица, которая использовалась при предыдущем вызове *объект-спейса*:format(*оператор-формата*). Например, после `box.space.tester:format({{'x','scalar'}})`, `box.space.tester:format()` вернет `[{'name': 'x', 'type': 'scalar'}]`.

Formatting or reformatting a large space will cause occasional *yields* so that other requests will not be blocked. If the other requests cause an illegal situation such as a field value of the wrong type, the formatting or reformatting will fail.

**Note regarding storage engine:** vinyl supports formatting of non-empty spaces. Primary index definition cannot be formatted.

## 5.10 space_object:frommap()

`object space_object`

`space_object:frommap(`*map*[, *option*]`)`

Конвертация ассоциативного массива в экземпляр кортежа или в таблицу. Ассоциативный массив должен состоять из пар «имя поля = значение». Имена полей и типы значений должны соответствовать именам и типам, ранее заданным для спейса с помощью *space_object:format()*.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*
- `map` (`field-value-pairs`) – ряд пар «поле = значение» в любом порядке.
- `option` (`boolean`) – единственный возможный параметр `{table = true|false}`; если параметр не указан, или же `{table = false}`, то возвращается „cdata“ (то есть кортеж); если `{table = true}`, то возвращается таблица.

**возвращает** кортеж или таблица.

**тип возвращаемого значения** кортеж или таблица

**Возможные ошибки:** отсутствует объект спейса `space_object`, или в спейсе нет формата; «unknown field» (неизвестное поле).

**Пример:**

```
-- Создание формата с двумя полями под названиями 'a' и 'b'.
-- Создание спейса с таким форматом.
-- Создание кортежа на основе ассоциативного массива по данному спейсу.
-- Создание таблицы на основе ассоциативного массива по данному спейсу.
tarantool> format1 = {{name='a',type='unsigned'},{name='b',type='scalar'}}
---
...
tarantool> s = box.schema.create_space('test', {format = format1})
---
...
tarantool> s:frommap({b = 'x', a = 123456})
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
- [123456, 'x']
...
tarantool> s:frommap({b = 'x', a = 123456}, {table = true})
---
- - 123456
  - x
...
```

## 5.11 space_object:get()

`object space_object`

`space_object:get(`*key*`)`

Поиск кортежа в данном спейсе.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*
- `key` (`scalar/table`) – значение должно совпасть с индексным ключом, который может быть составным.

**возвращает** кортеж, ключ индекса в котором совпадает с `key` или `nil`.

**тип возвращаемого значения** tuple

**Возможные ошибки:**

- `space_object` не существует.
- `ER_TRANSACTION_CONFLICT`, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Факторы сложности** Размер индекса, тип индекса, количество индексов, к которым получен доступ, настройки журнала упреждающей записи (WAL).

Функция `box.space...select` вернет набор кортежей в виде Lua-таблицы; функция `box.space...get` вернет самое большее один кортеж. Можно получить первый кортеж в спейсе, добавив `[1]`. Таким образом, `box.space.tester:get{1}` эквивалентна `box.space.tester:select{1}[1]`, если найден только один кортеж.

**Пример:**

```
box.space.tester:get{1}
```

**Использование имен полей вместо номеров полей:** в `get()` можно использовать имена полей, описанные в необязательном операторе *space_object:format()*. Это связано с тем, что функция `get()` возвращает объет, который можно использовать с большинством функций, описанных в *Вложенный модуль box.tuple*, включая *tuple_object[field-name]*.

Например, может форматировать спейс `tester` с полем под названием `x` и использовать имя `x` в определении индекса:

```
box.space.tester:format({{name='x',type='scalar'}})
box.space.tester:create_index('I',{parts={'x'}})
```

Тогда если `get` или `select` вернут отдельный кортеж, можно сослаться на поле „x“ в кортеже по имени:

```
box.space.tester:get{1}['x']
box.space.tester:select{1}[1]['x']
```

## 5.12 space_object:insert()

`object space_object`

`space_object:insert(`*tuple*`)`
Вставка кортежа в спейс.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*
- `tuple` (tuple/table) – вставляемый кортеж.

**возвращает** вставленный кортеж

**тип возвращаемого значения** tuple

**Возможные ошибки:**

- `ER_TUPLE_FOUND`, если уже существует кортеж с тем же уникальным значением ключа.
- `ER_TRANSACTION_CONFLICT`, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Пример:**

```
tarantool> box.space.tester:insert{5000,'tuple number five thousand'}
---
- [5000, 'tuple number five thousand']
...
```

Для получения дополнительной информации о сценариях использования и типичных ошибках, см. *Пример: использование операций с данными* далее в разделе.

## 5.13 space_object:len()

`object space_object`

`space_object:len()`
Возврат количества кортежей в спейсе. Если сравнивать с *count()*, то `len()` работает быстрее, поскольку не сканирует весь спейс для подсчета кортежей.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*

**возвращает** Количество кортежей в спейсе.

**Возможные ошибки:**

- ER_TRANSACTION_CONFLICT, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Пример:**

```
tarantool> box.space.tester:len()
---
- 2
...
```

**Note regarding storage engine:** vinyl supports `len()` but the result may be approximate. If an exact result is necessary then use *count()* or *pairs():length()*.

## 5.14 space_object:on_replace()

`object space_object`

`space_object:on_replace(`[*trigger-function*[, *old-trigger-function* ] ])

Создание «*триггера* замены». Функция-триггер `trigger-function` будет выполняться в случае операции `replace()` или `insert()`, или `update()`, или `upsert()`, или `delete()` над кортежем в спейсе `<space-name>`.

**Параметры**

- `trigger-function` (`function`) – функция, в которой будет триггер; подробнее о параметрах функции с триггером см. пример 2 ниже
- `old-trigger-function` (`function`) – существующая функция-триггер, которую заменит новая `trigger-function`

**возвращает** nil или указатель функции

Если указаны параметры `(nil, old-trigger-function)`, то старый триггер будет удален.

Если не указан ни один параметр, ответом будет список существующих функций с триггером.

Следует знать, что если активация триггера произошла в случае репликации или определенного вида подключения, функция может ссылаться на *box.session.type()*.

Подробная информация о характеристиках триггера находится в разделе *Триггеры*.

См. также *space_object:before_replace()*.

**Пример 1:**

```
tarantool> x = 0
         > function f ()
         >   x = x + 1
         > end
tarantool> box.space.my_space_name:on_replace(f)
```

**Пример 2:**

В функции с триггером может быть до 4 параметров:

- (кортеж) старое значение до начала запроса,
- (кортеж) новое значение после окончания выполнения запроса,
- (строка) имя спейса,

- (строка) тип запроса: `INSERT` (вставка), `DELETE` (удаление), `UPDATE` (обновление) или `REPLACE` (замена).

Например, следующий код вызывает вывод `nil` и `INSERT` при обработке запроса на вставку и вывод `[1, 'Hi']` и `DELETE` при обработке запроса на удаление:

```
box.schema.space.create('space_1')
box.space.space_1:create_index('space_1_index',{})
function on_replace_function (old, new, s, op) print(old) print(op) end
box.space.space_1:on_replace(on_replace_function)
box.space.space_1:insert{1,'Hi'}
box.space.space_1:delete{1}
```

**Пример 3:**

Следующая серия запросов создаст спейс, создаст индекс, создаст функцию, которая увеличит содержимое счетчика, создаст триггер, сделает две вставки, удалит спейс и отобразит значение счетчика – 2, поскольку функция выполняется однократно после каждой вставки.

```
tarantool> s = box.schema.space.create('space53')
tarantool> s:create_index('primary', {parts = {{field = 1, type = 'unsigned'}}})
tarantool> function replace_trigger()
         >   replace_counter = replace_counter + 1
         > end
tarantool> s:on_replace(replace_trigger)
tarantool> replace_counter = 0
tarantool> t = s:insert{1, 'First replace'}
tarantool> t = s:insert{2, 'Second replace'}
tarantool> s:drop()
tarantool> replace_counter
```

---

**Примечание:**

Так как всё, что выполняется в триггерах, уже находится в транзакции, не следует использовать в триггерных функциях `on_replace` и `before_replace`

- транзакции,
- yield-operations (*explicit* or not),
- действия, которые не разрешено использовать в транзакциях (см. *правило №2*).

**Пример:**

---

```
tarantool> box.space.test:on_replace(fiber.yield)
tarantool> box.space.test:replace{1, 2, 3}
2020-02-02 21:22:03.073 [73185] main/102/init.lua txn.c:532 E> ER_TRANSACTION_
↪YIELD: Transaction has been aborted by a fiber yield
---
- error: Transaction has been aborted by a fiber yield
...
```

## 5.15 space_object:before_replace()

object space_object

space_object:before_replace([*trigger-function*[, *old-trigger-function*]])

Создание «*триггера* замены». Функция-триггер `trigger-function` будет выполняться в случае операции `replace()` или `insert()`, или `update()`, или `upsert()`, или `delete()` над кортежем в спейсе `<space-name>`.

**Параметры**

- `trigger-function` (`function`) – функция, которая станет функцией-триггером; описание необязательных параметров для функций-триггеров см. в *on_replace*.
- `old-trigger-function` (`function`) – существующая функция-триггер, которую заменит `trigger-function`

**возвращает** nil или указатель функции

Если указаны параметры `(nil, old-trigger-function)`, старый триггер будет удален.

Если не указан ни один параметр, ответом будет список существующих функций-триггеров.

Следует знать, что если активация триггера произошла в случае репликации или определенного вида подключения, функция может ссылаться на *box.session.type()*.

Подробная информация о характеристиках триггера находится в разделе *Триггеры*.

См. также *space_object:on_replace()*.

Администраторы могут создавать триггеры замены с условием после замены `on_replace()` или до замены `before_replace()`. Если созданы оба типа, то все триггеры до замены `before_replace` выполняются до всех триггеров после замены `on_replace`. Функции для обоих типов триггеров `on_replace` и `before_replace` могут вносить изменения в базу данных, но только функции с триггерами до замены `before_replace` могут изменять кортеж, который будет заменен.

Поскольку функция-триггер до замены `before_replace` может вносить дополнительные изменения в старый кортеж, для нее также потребуются дополнительные ресурсы для вызова старого кортежа до внесения изменений. Таким образом, лучше использовать триггер после замены `on_replace`, если нет необходимости изменять старый кортеж. Тем не менее, это применимо только к движку memtx – что касается движка vinyl, такой вызов произойдет для любого типа триггера. (В memtx'е данные кортежа хранятся вместе с ключом индекса, поэтому нет необходимости в дополнительном поиске; для vinyl'а дело обстоит иначе, поэтому нужен дополнительный поиск.)

Если нет необходимости в дополнительных изменениях, следует использовать `on_replace` вместо `before_replace`. Как правило, `before_replace` используется только для определенных сценариев репликации – в части разрешения конфликтов.

Что случится после возврата значения, которое может вернуть функция-триггер `before_replace`, зависит от этого значения. А именно:

- если нет возвращаемого значения, выполнение продолжается со вставкой|заменой нового значения;
- если значение – nil, то кортеж будет удален;
- если значение совпадает со старым, то функция `on_replace` не вызывается, и изменение данных не происходит. Возвращаемого значения в таком случае не будет.

- если значение совпадает с новым, то считаем, что вызова функции `before_replace` не было;
- если значение другое, выполнение продолжается со вставкой/заменой нового значения.

Тем не менее, если функция-триггер возвращает старый кортеж или вызовет *run_triggers(false)*, это не повлияет на другие триггеры, активируемые в том же запросе вставки, обновления или замены.

**Пример:**

Далее представлены функции `before_replace`: не возвращает значение, возвращает nil, возвращает совпадающее со старым значение, возвращает совпадающее с новым значение, возвращает другое значение.

```
function f1 (old, new) return end
function f2 (old, new) return nil end
function f3 (old, new) return old end
function f4 (old, new) return new end
function f5 (old, new) return box.tuple.new({new[1],'b'}) end
```

## 5.16 space_object:pairs()

`object space_object`

`space_object:pairs(`[*key*[, *iterator*]]`)`

Search for a tuple or a set of tuples in the given space, and allow iterating over one tuple at a time. To search by the specific index, use the *index_object:pairs()* method.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*
- `key` (`scalar/table`) – значение должно совпасть с индексным ключом, который может быть составным
- `iterator` – the *iterator type*. The default iterator type is „EQ“
- `after` – a tuple or the position of a tuple (*tuple_pos*) after which `pairs` starts the search. You can pass an empty string or *box.NULL* to this option to start the search from the first tuple.

**возвращает** The iterator, which can be used in a `for/end` loop or with totable().

**Возможные ошибки:**

- no such space
- wrong type
- `ER_TRANSACTION_CONFLICT` if a transaction conflict is detected in the *MVCC transaction mode*
- iterator position is invalid

**Факторы сложности:** Размер индекса, тип индекса.

For information about iterators“ internal structures, see the «Lua Functional library» documentation.

**Examples:**

Below are few examples of using `pairs` with different parameters. To try out these examples, you need to bootstrap a Tarantool instance as described in *Using data operations*.

```
-- Insert test data --
tarantool> bands:insert{1, 'Roxette', 1986}
           bands:insert{2, 'Scorpions', 1965}
           bands:insert{3, 'Ace of Base', 1987}
           bands:insert{4, 'The Beatles', 1960}
           bands:insert{5, 'Pink Floyd', 1965}
           bands:insert{6, 'The Rolling Stones', 1962}
           bands:insert{7, 'The Doors', 1965}
           bands:insert{8, 'Nirvana', 1987}
           bands:insert{9, 'Led Zeppelin', 1968}
           bands:insert{10, 'Queen', 1970}
---
...

-- Select all tuples by the primary index --
tarantool> for _, tuple in bands:pairs() do
               print(tuple)
           end
[1, 'Roxette', 1986]
[2, 'Scorpions', 1965]
[3, 'Ace of Base', 1987]
[4, 'The Beatles', 1960]
[5, 'Pink Floyd', 1965]
[6, 'The Rolling Stones', 1962]
[7, 'The Doors', 1965]
[8, 'Nirvana', 1987]
[9, 'Led Zeppelin', 1968]
[10, 'Queen', 1970]
---
...

-- Select all tuples whose primary key values are between 3 and 6 --
tarantool> for _, tuple in bands:pairs(3, {iterator = "GE"}) do
             if (tuple[1] > 6) then break end
             print(tuple)
           end
[3, 'Ace of Base', 1987]
[4, 'The Beatles', 1960]
[5, 'Pink Floyd', 1965]
[6, 'The Rolling Stones', 1962]
---
...

-- Select all tuples after the specified tuple --
tarantool> for _, tuple in bands:pairs({}, {after={7, 'The Doors', 1965}}) do
               print(tuple)
           end
[8, 'Nirvana', 1987]
[9, 'Led Zeppelin', 1968]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
[10, 'Queen', 1970]
---
...
```

## 5.17 space_object:put()

См. *space_object:replace() / put()*.

## 5.18 space_object:rename()

`object space_object`

`space_object:rename(`*space-name*`)`
Переименование спейса.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*
- `space-name` (*string*) – новое имя спейса

**возвращает** nil

**Возможные ошибки:** `space_object` не существует

**Пример:**

```
tarantool> box.space.space55:rename('space56')
---
...
tarantool> box.space.space56:rename('space55')
---
...
```

## 5.19 space_object:replace() / put()

`object space_object`

`space_object:replace(`*tuple*`)`
`space_object:put(`*tuple*`)`
Вставка кортежа в спейс. Если уже существует кортеж с тем же первичным ключом, `box.space...:replace()` заменит существующий кортеж новым. Варианты синтаксиса (`box.space...:replace()` и `box.space...:put()`) приведут к одному результату, но последний иногда используется как противоположность `box.space...:get()`.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*
- `tuple` (`table/tuple`) – вставляемый кортеж

**возвращает** вставленный кортеж

**тип возвращаемого значения** tuple

**Возможные ошибки:**

- ER_TUPLE_FOUND, если уже существует другой кортеж с тем же уникальным значением ключа (это произойдет только в том случае, если есть уникальный вторичный индекс).
- ER_TRANSACTION_CONFLICT, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Факторы сложности:** Размер индекса, тип индекса, количество индексов, к которым получен доступ, настройки журнала упреждающей записи (WAL).

**Пример:**

```
box.space.tester:replace{5000, 'tuple number five thousand'}
```

Для получения дополнительной информации о сценариях использования и типичных ошибках, см. *Пример: использование операций с данными* далее в разделе.

## 5.20 space_object:run_triggers()

```
object space_object
```

space_object:run_triggers(*true/false*)

В тот момент, когда *триггер* определен, он активируется автоматически, то есть он будет исполняться. Триггеры *replace* можно отключить с помощью box.space.*space-name*:run_triggers(false) и повторно активировать с помощью box.space.*space-name*:run_triggers(true).

**возвращает** nil

**Пример:**

Следующая серия запросов ассоциирует существующую функцию с именем F с существующим спейсом с именем T, ассоциирует функцию во второй раз с тем же спейсом (чтобы вызвать ее дважды), отключит все триггеры на T и удалит каждый триггер, заменив его на nil.

```
tarantool> box.space.T:on_replace(F)
tarantool> box.space.T:on_replace(F)
tarantool> box.space.T:run_triggers(false)
tarantool> box.space.T:on_replace(nil, F)
tarantool> box.space.T:on_replace(nil, F)
```

## 5.21 space_object:select()

`object space_object`

`space_object:select(`[*key*[, *options* ]])

Search for a tuple or a set of tuples in the given space by the primary key. To search by the specific index, use the *index_object:select()* method.

---

**Примечание:** Note that this method doesn't yield. For details, see *Cooperative multitasking*.

---

**Параметры**

- `space_object` (*space_object*) – an *object reference*.
- `key` (`scalar/table`) – a value to be matched against the index key, which may be multi-part.
- `options` (`table/nil`) – none, any, or all of the same options that *index_object:select()* allows:
  - `options.iterator` – the *iterator type*. The default iterator type is „EQ“.
  - `options.limit` – the maximum number of tuples.
  - `options.offset` – the number of tuples to skip.
  - `options.after` – a tuple or the position of a tuple (*tuple_pos*) after which `select` starts the search. You can pass an empty string or *box.NULL* to this option to start the search from the first tuple.
  - `options.fetch_pos` – if **true**, the `select` method returns the position of the last selected tuple as the second value.

    ---

    **Примечание:** The `after` and `fetch_pos` options are supported for the `TREE` *index* only.

    ---

**возвращает** This function might return one or two values:

- The tuples whose primary-key fields are equal to the fields of the passed key. If the number of passed fields is less than the number of fields in the primary key, then only the passed fields are compared, so `select{1,2}` matches a tuple whose primary key is `{1,2,3}`.
- (Optionally) If `options.fetch_pos` is set to **true**, returns a base64-encoded string representing the position of the last selected tuple as the second value. If no tuples are fetched, returns `nil`.

**тип возвращаемого значения**

- массив кортежей
- (Optionally) string

**Возможные ошибки:**

- no such space
- wrong type

- `ER_TRANSACTION_CONFLICT` if a transaction conflict is detected in the *MVCC transaction mode*
- iterator position is invalid

**Факторы сложности:** Размер индекса, тип индекса

**Examples:**

Below are few examples of using `select` with different parameters. To try out these examples, you need to bootstrap a Tarantool instance as described in *Using data operations*.

```
-- Insert test data --
tarantool> bands:insert{1, 'Roxette', 1986}
           bands:insert{2, 'Scorpions', 1965}
           bands:insert{3, 'Ace of Base', 1987}
           bands:insert{4, 'The Beatles', 1960}
           bands:insert{5, 'Pink Floyd', 1965}
           bands:insert{6, 'The Rolling Stones', 1962}
           bands:insert{7, 'The Doors', 1965}
           bands:insert{8, 'Nirvana', 1987}
           bands:insert{9, 'Led Zeppelin', 1968}
           bands:insert{10, 'Queen', 1970}
---
...

-- Select a tuple by the specified primary key --
tarantool> bands:select(4)
---
- - [4, 'The Beatles', 1960]
...

-- Select maximum 3 tuples with the primary key value greater than 3 --
tarantool> bands:select({3}, {iterator='GT', limit = 3})
---
- - [4, 'The Beatles', 1960]
  - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
...

-- Select maximum 3 tuples after the specified tuple --
tarantool> bands:select({}, {after = {4, 'The Beatles', 1960}, limit = 3})
---
- - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
  - [7, 'The Doors', 1965]
...

-- Select first 3 tuples and fetch a last tuple's position --
tarantool> result, position = bands:select({}, {limit = 3, fetch_pos = true})
---
...
-- Then, pass this position as the 'after' parameter --
tarantool> bands:select({}, {limit = 3, after = position})
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- - [4, 'The Beatles', 1960]
  - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
...
```

---

**Примечание:** You can get a field from a tuple both by the field number and field name. See example: *using field names instead of field numbers*.

---

## 5.22 space_object:truncate()

`object space_object`

`space_object:truncate()`

Удаление всех кортежей. Метод выполняется в фоновом режиме и не блокирует последующие запросы.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*

**Факторы сложности:** Размер индекса, тип индекса, количество кортежей, к которым получен доступ.

**возвращает** nil

Метод `truncate` может вызвать только тот пользователь, который создал спейс, или другой пользователь через функцию `setuid`, созданную пользователем, который создал спейс. Более подробную информацию о функциях `setuid` можно получить в справочнике по for *box.schema.func.create()*.

---

**Примечание:** В версиях Tarantool старше v. 2.10.0 не следует вызывать `truncate` из транзакции. Подробности в тикете gh-6123.

---

**Пример:**

```
tarantool> box.space.tester:truncate()
---
...
tarantool> box.space.tester:len()
---
- 0
...
```

## 5.23 space_object:update()

object space_object

space_object:update(*key*, *{{operator*, *field_identifier*, *value}*, *...}*)

Обновление кортежа.

Функция update поддерживает операции над полями — присваивание, арифметические операции (если поле числовое), вырезание и вставку фрагментов поля, удаление или вставку поля. Несколько операций можно объединить в отдельный запрос обновления, и в таком случае они будут выполняться атомарно и последовательно. Для каждой операции необходимо указать идентификатор поля, как правило числовой. При выполнении нескольких операций номер поля для каждой операции принимается относительно последнего состояния кортежа, то есть как если бы уже были выполнены все предыдущие операции в обновлении с несколькими операциями. Другими словами, всегда лучше объединить несколько вызовов update в один без изменений семантики.

Возможные операторы:

- + для сложения; значения должны быть числовыми, например, unsigned или decimal
- - для вычитания; значения должны быть числовыми
- & для операции побитового И; значения должны быть числовыми типа unsigned
- | для операции побитового ИЛИ; значения должны быть числовыми типа unsigned
- ^ для операции побитового исключающего ИЛИ; значения должны быть числовыми типа unsigned
- : для разделения строк.
- ! для вставки нового поля.
- # для удаления.
- = для присваивания.

Возможные операторы:

- Положительный номер позиции поля. Первое поле — 1, второе — 2 и так далее.
- Отрицательный номер позиции поля. Последнее поле — 1, предпоследнее — 2 и так далее. Другими словами: (#tuple + отрицательный номер позиции поля + 1).
- Имя. Если спейс был отформатирован с помощью *space_object:format()*, здесь может быть строка с именем поля.

**Параметры**

- space_object (*space_object*) – *ссылка на объект*
- key (scalar/table) – значения поля первичного ключа, которые должны возвращаться в виде Lua-таблицы, если ключ составной
- operator (*string*) – тип операции, представленный строкой
- field_identifier (number-or-string) – к какому полю применяется операция.
- value (lua_value) – какое значение применяется

**возвращает**

- обновленный кортеж
- nil, если ключ не найден

**тип возвращаемого значения** кортеж или nil

**Возможные ошибки:**

- Нельзя изменять поле первичного ключа.
- ER_TRANSACTION_CONFLICT, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Факторы сложности** Размер индекса, тип индекса, количество индексов, к которым получен доступ, настройки журнала упреждающей записи (WAL).

Таким образом, в инструкции:

```
s:update(44, {{'+', 1, 55 }, {'=', 3, 'x'}})
```

значение первичного ключа равно 44, заданы операторы '+' и '=', что означает *прибавление значение к полю, а затем присваивание значения полю*, первое затронутое поле – это поле 1, к нему прибавляется значение 55, второе затронутое поле – это поле 3, ему присваивается значение 'x'.

**Пример:**

Предположим, что изначально есть спейс под названием tester с первичным индексом, тип которого – unsigned. Есть один кортеж с полем №1 field[1] = 999 и полем №2 field[2] = 'A'.

В обновлении: box.space.tester:update(999, {{'=', 2, 'B'}}) Первый аргумент – это tester, то есть обновление происходит в спейсе tester. Второй аргумент – 999, то есть затронутый кортеж определяется по значению первичного ключа = 999. Третий аргумент – =, то есть будет одна операция – *присваивание полю*. Четвертый аргумент – 2, то есть будет затронуто поле №2 field[2]. Пятый аргумент – 'B', то есть содержимое field[2] изменится на 'B'. Таким образом, после данного обновления field[1] = 999, а field[2] = 'B'.

В обновлении: box.space.tester:update({999}, {{'=', 2, 'B'}}) Аргументы повторяются за исключением того, что ключ передается в виде Lua-таблицы (в фигурных скобках). В этом нет необходимости, если первичный ключ содержит только одно поле, но было бы необходимо, если бы в первичном ключе было больше одного поля. Таким образом, после данного обновления field[1] = 999, а field[2] = 'B' (без изменений).

В обновлении: box.space.tester:update({999}, {{'=', 3, 1}}) Аргументы повторяются за исключением того, что четвертым аргументом будет 3, то есть будет затронуто поле №3 field[3]. Ничего страшного, что до этого поле field[3] не существовало. Оно добавится. Таким образом, после данного обновления field[1] = 999, field[2] = 'B', field[3] = 1.

В обновлении: box.space.tester:update({999}, {{'+', 3, 1}}) Аргументы повторяются за исключением того, что третьим аргументом будет '+', то есть будет операция добавления, а не присваивания. Поскольку``field[3]`` ранее содержало значение 1, это означает, что к 1 прибавится 1. Таким образом, после данного обновления field[1] = 999, field[2] = 'B', field[3] = 2.

В обновлении: box.space.tester:update({999}, {{'|', 3, 1}, {'=', 2, 'C'}}) Основная идея состоит в том, чтобы изменить одновременно два поля. Форматами будут '|' и =, то есть имеем две операции: ИЛИ и присваивание. Четвертый и пятый аргументы означают, что над полем field[3] проводится операция ИЛИ со значением 1. Седьмой и восьмой

аргументы означают, что полю `field[2]` присваивается `'C'`. Таким образом, после данного обновления `field[1]` = 999, `field[2]` = `'C'`, `field[3]` = 3.

В обновлении: `box.space.tester:update({999}, {{'#', 2, 1}, {'-', 2, 3}})` Основная идея состоит в том, чтобы удалить поле `field[2]`, а затем вычесть 3 из `field[3]`. Но после удаления, произойдет перенумерация, поэтому поле `field[3]` становится `field[2]` до того, как мы вычтем из него 3, вот почему седьмым аргументом будет 2, а не 3. Таким образом, после данного обновления `field[1]` = 999, `field[2]` = 0.

В обновлении: `box.space.tester:update({999}, {{'=', 2, 'XYZ'}})` Создаем длинную строку, чтобы в следующем примере сработало разделение. Таким образом, после данного обновления `field[1]` = 999, `field[2]` = `'XYZ'`.

В обновлении: `box.space.tester:update({999}, {{':', 2, 2, 1, '!!'}})` Третьим аргументом будет `':'`, то есть это пример разделения. Четвертым аргументом будет 2, поскольку изменение произойдет в поле `field[2]`. Пятым аргументом будет 2, поскольку удаление начнется со второго байта. Шестым аргументом будет 1, количество удаляемых байтов – 1. Седьмым аргументом будет `'!!'`, поскольку в данном положении будет добавляться `'!!'`. Таким образом, после данного обновления `field[1]` = 999, `field[2]` = `'X!!Z'`.

Для получения дополнительной информации о сценариях использования и типичных ошибках, см. *Пример: использование операций с данными* далее в разделе.

Начиная с версии Tarantool 2.3 кортеж можно обновить с помощью *JSON-путей*.

## 5.24 space_object:upsert()

`object space_object`

`space_object:upsert(`*{tuple}*, *{{operator*, *field_identifier*, *value}*, *...}*`)`

Обновить или вставить кортеж.

Если уже существует кортеж, который совпадает с ключевыми полями `tuple`, запрос приведет к тому же результату, что и *space_object:update()*, и используется параметр `{{operator, field_identifier, value}, ...}`. Если нет кортежа, который совпадает с ключевыми полями `tuple`, запрос приведет к тому же результату, что и *space_object:insert()*, и используется параметр `{tuple}`. Однако, в отличие от `insert` или `update`, `upsert` не считывает кортеж и не выполняет проверку на ошибки перед возвратом — это функциональная особенность, которая повышает производительность, но требует большей осторожности от пользователя.

**Параметры**

- `space_object` (*space_object*) – *ссылка на объект*
- `tuple` (`table/tuple`) – вставляемый по умолчанию кортеж, если не найдет аналог
- `operator` (*string*) – тип операции, представленный строкой
- `field_identifier` (`number`) – к какому полю применяется операция
- `value` (`lua_value`) – какое значение применяется

**возвращает** null

**Возможные ошибки:**

- Нельзя изменять поле первичного ключа.

- Нельзя проводить операцию upsert в спейсе, в котором есть уникальный вторичный индекс.
- `ER_TRANSACTION_CONFLICT`, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Факторы сложности:** Размер индекса, тип индекса, количество индексов, к которым получен доступ, настройки журнала упреждающей записи (WAL).

**Пример:**

```
box.space.tester:upsert({12,'c'}, {{'=', 3, 'a'}, {'=', 4, 'b'}})
```

Для получения дополнительной информации о сценариях использования и типичных ошибках, см. *Пример: использование операций с данными* далее в разделе.

## 5.25 space_object extensions

You can extend `space_object` with custom functions as follows:

1. Create a Lua function.
2. Add the function name to a predefined global variable `box.schema.space_mt`, which has the `table` type. Adding to `box.schema.space_mt` makes the function available for all spaces.
3. Call the function on the `space_object`: `space_object:function-name([parameters])`.

Alternatively, you can make a user-defined function available for only one space by calling `getmetatable(space_object)` and then adding the function name to the meta table.

See also: *index_object extensions*.

**Пример:**

```
-- Доступный для любого спейса, без параметров.
-- После таких запросов значение глобальной переменной global_variable будет 6.
box.schema.space.create('t')
box.space.t:create_index('i')
global_variable = 5
function f(space_arg) global_variable = global_variable + 1 end
box.schema.space_mt.counter = f
box.space.t:counter()
```

## 5.26 box.space.create_check_constraint()

**Предупреждение:** This function was removed in *2.11.0*. The check constraint mechanism is replaced with the new tuple constraints. Learn more about tuple constraints in *Constraints*.

`object space_object`

space_object:create_check_constraint(*check_ constraint_ name*, *expression*)

Создание проверочного ограничения. Проверочное ограничение — это требование, которое должно соблюдаться при вставке или обновлении кортежа в спейсе. Проверочные ограничения, созданные с помощью space_object:create_check_constraint действуют так же, как проверочные ограничения, которые были созданы при помощи оператора SQL CHECK() в *выражении CREATE TABLE*.

**Параметры**

- space_object (*space_object*) – *ссылка на объект*
- check_constraint_name (*string*) – имя проверочного ограничения, которое должно соответствовать *правилам именования объектов*
- expression (*string*) – SQL-код выражения, которое должно возвращать результат типа boolean

**возвращает** объект проверочного ограничения

**тип возвращаемого значения** check_constraint_object

Спейс должен быть форматирован с помощью *space_ object:format()*, чтобы выражение могло содержать имена полей. Спейс должен быть пустым. Спейс не должен быть системным.

Выражение должно возвращать true или false и должно быть детерминированным. Это может быть любое SQL-выражение (не Lua), которое содержит имена полей, встроенные имена функций, литералы и операторы. Не подзапросы. Если имя поля содержит символы нижнего регистра, его следует заключить в «двойные кавычки».

Проверка ограничений производится перед выполнением запроса, одновременно с Lua-триггерами *before_ replace*. При наличии двух и более проверочных ограничений или триггеров before_replace они упорядочиваются по времени создания. (Это изменение предыдущего поведения проверочных ограничений, которое вызывало проверку до формирования кортежа).

Проверочные ограничения можно удалить с помощью *space_object*.ck_constraint.*check_constraint_name*:drop().

Проверочные ограничения можно отключить с помощью *space_object*.ck_constraint.*check_constraint_name*:enable(false) или *check_constraint_object*:enable(false). Проверочные ограничения можно включить с помощью *space_object*.ck_constraint.*check_constraint_name*:enable(true) или *check_constraint_object*:enable(true). По умолчанию, проверочное ограничение включено, то есть при выполнении запроса производится проверка, но его можно отключить, то есть проверка производиться не будет.

В процессе восстановления, например при запуске Tarantool-сервера, проверка производится, только если указан параметр *force_ recovery*.

**Пример:**

```
box.schema.space.create('t')
box.space.t:format({{name = 'f1', type = 'unsigned'},
                    {name = 'f2', type = 'string'},
                    {name = 'f3', type = 'string'}})
box.space.t:create_index('i')
box.space.t:create_check_constraint('c1', [["f2" > 'A']])
box.space.t:create_check_constraint('c2',
                        [["f2"=UPPER("f3") AND NOT "f2" LIKE '__']])
-- Этот запрос вставки не выполнится, выражение с ограничением c1 возвращает␣
↪false
```

(continues on next page)

(продолжение с предыдущей страницы)

```
box.space.t:insert{1, 'A', 'A'}
-- Этот запрос вставки не выполнится, выражение с ограничением c2 возвращает
↪false
box.space.t:insert{1, 'B', 'c'}
-- Этот запрос вставки выполнится, оба выражения с ограничениями возвращают true
box.space.t:insert{1, 'B', 'b'}
-- Этот запрос обновления не выполнится, выражение с ограничением c2 возвращает
↪false
box.space.t:update(1, {{'=', 2, 'xx'}, {'=', 3, 'xx'}})
```

Список проверочных ограничений представлен в *box.space._ck_constraint*.

## 5.27 space_object:enabled

`object space_object`

`space_object.enabled`
Определение активности спейса. Значение `false` указывает на отсутствие индекса.

## 5.28 space_object:field_count

`object space_object`

`space_object.field_count`
Необходимость подсчета полей всех кортежей в спейсе, который можно изначально задать следующим образом:

```
box.schema.space.create(..., {
    ... ,
    field_count = *field_count_value* ,
    ...
})
```

По умолчанию, будет использоваться значение 0, что указывает на отсутствие необходимости подсчета полей.

**Пример:**

```
tarantool> box.space.tester.field_count
---
- 0
...
```

## 5.29 space_object.id

object space_object

space_object.id

Порядковый номер спейса. На спейс можно ссылаться либо по имени, либо по номеру. Таким образом, если идентификатором спейса `tester` будет `id = 800`, то `box.space.tester:insert{0}` и `box.space[800]:insert{0}` представляют собой равнозначные запросы.

**Пример:**

```
tarantool> box.space.tester.id
---
- 512
...
```

## 5.30 space_object.index

object space_object

index

Контейнер для всех определенных индексов. Есть Lua-объект типа *box.index* с методами поиска кортежей и итерации по ним в заданном порядке.

Чтобы сбросить, use *box.stat.reset()*.

**тип возвращаемого значения** таблица

**Пример:**

```
-- проверка количества индексов для спейса 'tester'
tarantool> local counter=0; for i=0,#box.space.tester.index do
  if box.space.tester.index[i]~=nil then counter=counter+1 end
  end; print(counter)
1
---
...
-- проверка, что тип индекса -- первичный 'primary'
tarantool> box.space.tester.index.primary.type
---
- TREE
...
```

## 5.31 box.space._cluster

box.space._cluster
: _cluster – это системный спейс для поддержки *функции репликации*.

## 5.32 box.space._func

box.space._func
: A system space containing functions created using *box.schema.func.create()*. If a function's definition is specified in the *body* option, this function is *persistent*. In this case, its definition is stored in a snapshot and can be recovered if the server restarts.

---

**Примечание:** *Представление системного спейса* _func — _vfunc.

---

## 5.33 box.space._index

box.space._index
: _index – это системный спейс.

Кортежи в данном спейсе включают в себя следующие поля:

- id (= идентификатор спейса),
- iid (= номер индекса в спейсе),
- name,
- type,
- opts (например, уникальная опция), [tuple-field-no, tuple-field-type ...].

Вот что при обычной установке включает в себя спейс _index:

```
tarantool> box.space._index:select{}
---
- - [272, 0, 'primary', 'tree', {'unique': true}, [[0, 'string']]]
  - [280, 0, 'primary', 'tree', {'unique': true}, [[0, 'unsigned']]]
  - [280, 1, 'owner', 'tree', {'unique': false}, [[1, 'unsigned']]]
  - [280, 2, 'name', 'tree', {'unique': true}, [[2, 'string']]]
  - [281, 0, 'primary', 'tree', {'unique': true}, [[0, 'unsigned']]]
  - [281, 1, 'owner', 'tree', {'unique': false}, [[1, 'unsigned']]]
  - [281, 2, 'name', 'tree', {'unique': true}, [[2, 'string']]]
  - [288, 0, 'primary', 'tree', {'unique': true}, [[0, 'unsigned'], [1, 'unsigned
↪']]]
  - [288, 2, 'name', 'tree', {'unique': true}, [[0, 'unsigned'], [2, 'string']]]
  - [289, 0, 'primary', 'tree', {'unique': true}, [[0, 'unsigned'], [1, 'unsigned
↪']]]
  - [289, 2, 'name', 'tree', {'unique': true}, [[0, 'unsigned'], [2, 'string']]]
  - [296, 0, 'primary', 'tree', {'unique': true}, [[0, 'unsigned']]]
  - [296, 1, 'owner', 'tree', {'unique': false}, [[1, 'unsigned']]]
  - [296, 2, 'name', 'tree', {'unique': true}, [[2, 'string']]]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...
```

*Представление системного спейса* `_index` — `_vindex`.

## 5.34 box.space._vindex

`box.space._vindex`
*Представление системного спейса* *_index* — `_vindex`.

## 5.35 box.space._priv

`box.space._priv`
`_priv` – это системный спейс, где хранятся *права*.

Кортежи в данном спейсе включают в себя следующие поля:

- числовой идентификатор пользователя, который выдал права («grantor_id»),
- числовой идентификатор пользователя, который получил права («grantee_id»),
- тип объекта: „space“, „index“, „function“, „sequence“, „user“, „role“, или „universe“,
- числовой идентификатор объекта,
- тип операции: «read» = 1, «write» = 2, «execute» = 4, «create» = 32, «drop» = 64, «alter» = 128, или их комбинация, например «read,write,execute».

Подробнее о правах пользователя см. в разделе *Управление доступом*.

*Представление системного спейса* `_priv` — `_vpriv`.

## 5.36 box.space._vpriv

`box.space._vpriv`
*Представление системного спейса* *_priv* — `_vpriv`.

## 5.37 box.space._schema

`box.space._schema`
`_schema` – это системный спейс.

Этот спейс включает в себя следующие кортежи:

- кортеж `version` с информацией о версии данного экземпляра Tarantool,
- кортеж `cluster` с идентификатором набора реплик данного экземпляра,
- кортеж `max_id` с максимальным ID спейса,

- кортежи `once...`, которые соответствуют определенным блокам *box.once()* из *файла инициализации* экземпляра. Первое поле в таких кортежах содержит значение ключа `key` из соответствующего блока `box.once()` с префиксом „once“ (например, `oncehello`), поэтому можно легко найти кортеж, который соответствует определенному блоку `box.once()`.

**Пример:**

Вот что при обычной установке включает в себя спейс `_schema` (обратите внимание на кортежи для двух блоков `box.once()`: `'oncebye'` и `'oncehello'`):

```
tarantool> box.space._schema:select{}
---
- - ['cluster', 'b4e15788-d962-4442-892e-d6c1dd5d13f2']
  - ['max_id', 512]
  - ['oncebye']
  - ['oncehello']
  - ['version', 1, 7, 2]
```

## 5.38 box.space._sequence

`box.space._sequence`

`_sequence` – это системный спейс для поддержки *последовательностей*. Он содержит персистентную информацию, определенную с помощью *box.schema.sequence.create()* или *sequence_object:alter()*.

*Представление системного спейса* `_sequence` — `_vsequence`.

## 5.39 box.space._sequence_data

`box.space._sequence_data`

`_sequence_data` – это системный спейс для поддержки *последовательностей*.

Каждый кортеж в спейсе `_sequence_data` содержит два поля:

- идентификатор последовательности и
- последнее значение, возвращенное генератором последовательностей (временная информация).

Не гарантируется, что этот спейс будет обновляться сразу же после каждого запроса на изменение данных.

## 5.40 box.space._space

`box.space._space`

`_space` – это системный спейс. Он содержит информацию о всех спейсах, хранящихся в данном экземпляре Tarantool - как системные, так и созданные пользователями.

Кортежи в данном спейсе включают в себя следующие поля:

- `id`,
- `owner` (= идентификатор пользователя, которому принадлежит спейс),

- `name`, `engine`, `field_count`,
- `flags` (например, временный),
- `format` (как задано с помощью *оператора формата*).

Эти поля задаются во время создания спейса с помощью *box.schema.space.create()*.

*Представление системного спейса* `_space` — `_vspace`.

**Пример №1:**

Следующая функция отобразит все простые поля во всех кортежах спейса `_space`.

```
function example()
  local ta = {}
  local i, line
  for k, v in box.space._space:pairs() do
    i = 1
    line = ''
    while i <= #v do
      if type(v[i]) ~= 'table' then
        line = line .. v[i] .. ' '
      end
    i = i + 1
    end
    table.insert(ta, line)
  end
  return ta
end
```

Вот что при обычной установке вернет `example()`:

```
tarantool> example()
---
- - '272 1 _schema memtx 0  '
  - '280 1 _space memtx 0  '
  - '281 1 _vspace sysview 0  '
  - '288 1 _index memtx 0  '
  - '296 1 _func memtx 0  '
  - '304 1 _user memtx 0  '
  - '305 1 _vuser sysview 0  '
  - '312 1 _priv memtx 0  '
  - '313 1 _vpriv sysview 0  '
  - '320 1 _cluster memtx 0  '
  - '512 1 tester memtx 0  '
  - '513 1 origin vinyl 0  '
  - '514 1 archive memtx 0  '
...
```

**Пример №2:**

Следующий набор запросов создаст спейс, используя `box.schema.space.create()` с *оператором формата*, затем выберет кортеж из `_space` для нового спейса. Этот пример иллюстрирует стандартное применение оператора `format` с использованием рекомендованных имен и типов данных для полей.

```
tarantool> box.schema.space.create('TM', {
         >   id = 12345,
         >   format = {
         >     [1] = {["name"] = "field_1"},
         >     [2] = {["type"] = "unsigned"}
         >   }
         > })
---
- index: []
  on_replace: 'function: 0x41c67338'
  temporary: false
  id: 12345
  engine: memtx
  enabled: false
  name: TM
  field_count: 0
- created
...
tarantool> box.space._space:select(12345)
---
- - [12345, 1, 'TM', 'memtx', 0, {}, [{'name': 'field_1'}, {'type': 'unsigned'}]]
...
```

## 5.41 box.space._vspace

box.space._vspace
*Представление системного спейса _space* — _vspace.

## 5.42 box.space._space_sequence

box.space._space_sequence
_space_sequence is a system space. It contains connections between spaces and sequences.

Tuples in this space contain the following fields:

- id (unsigned) – space id
- sequence_id (unsigned) – id of the attached sequence
- is_generated (boolean) – true if the sequence was created automatically via a space:create_index('pk', {sequence = true}) call
- field (unsigned) – id of the space field to which the sequence is attached.
- path (string) – path to the data within the field that is set using the attached sequence.

The *system space view* for _space_sequence is _vspace_sequence.

**Example**

```
-- Create a sequence --
box.schema.sequence.create('id_seq',{min=1000, start=1000})
-- Create a space --
```

(continues on next page)

(продолжение с предыдущей страницы)

```
box.schema.space.create('customers')

-- Create an index that uses the sequence --
box.space.customers:create_index('primary',{ sequence = 'id_seq' })

-- Create a space --
box.schema.space.create('orders')

-- Create an index that uses an auto sequence --
box.space.orders:create_index( 'primary', { sequence = true })

-- Check the connections between spaces and sequences
box.space._space_sequence:select{}
--[[
---
- - [512, 1, false, 0, '']
  - [513, 2, true, 0, '']
...
--]]
```

## 5.43 box.space._vspace_sequence

box.space._vspace_sequence

_vspace_sequence is the *system space view* for *_space*.

## 5.44 box.space._user

box.space._user

_user is a system space where user names and password hashes are stored. Learn more about Tarantool's access control system from the *Управление доступом* topic.

Кортежи в данном спейсе включают в себя следующие поля:

- a numeric id of the tuple («id»)
- a numeric id of the tuple's creator
- a name
- a type: „user“ or „role“
- (optional) a password hash
- (optional) an array of previous authentication data
- (optional) a timestamp of the last password update

В спейсе _user есть пять специальных кортежей: „guest“, „admin“, „public“, „replication“ и „super“.

| Имя | ID | Тип | Описание |
|---|---|---|---|
| guest | 0 | user (пользователь) | Default user when connecting remotely. Usually, an untrusted user with few privileges. |
| admin | 1 | user (пользователь) | Default user when using Tarantool as a console. Usually, an *administrative user* with all privileges. |
| public | 2 | роль | Заданная *роль*, которая автоматически выдается новым пользователям при их создании методом `box.schema.user.create(имя-пользователя)`. Таким образом, лучше всего выдать права на чтение „read“ спейса „t“ каждому когда-либо созданному пользователю с помощью `box.schema.role.grant('public','read','space','t')`. |
| replication | 3 | роль | Заданная *роль*, выдаваемая пользователем „admin“ другим пользователям для использования функций *репликации*. |
| super | 31 | роль | Заданная *роль*, выдаваемая пользователем „admin“ другим пользователям для получения всех прав на все объекты. Для роли „super“ такие права выданы на „universe“: чтение, запись, выполнение, создание, удаление, изменение. |

To select a tuple from the `_user` space, use `box.space._user:select()`. In the example below, `select` is executed for a user with id = 0. This is the „guest“ user that has no password.

```
tarantool> box.space._user:select{0}
---
- - [0, 1, 'guest', 'user']
...
```

**Предупреждение:** To change tuples in the `_user` space, do not use ordinary `box.space` functions for insert, update, or delete. Learn more from *Managing users*.

*Представление системного спейса* `_user` — `_vuser`.

## 5.45 box.space._vuser

box.space._vuser
*Представление системного спейса _user* — _vuser.

## 5.46 box.space._ck_constraint

box.space._ck_constraint
_ck_constraint — это системный спейс, в котором хранятся проверочные ограничения.

Кортежи в данном спейсе включают в себя следующие поля:

- числовой идентификатор спейса («space_id»),
- имя,
- откладывается ли проверка («is_deferred»),
- язык выражения, например „SQL“,
- выражение («code»).

**Пример:**

```
tarantool> box.space._ck_constraint:select()
---
- - [527, 'c1', false, 'SQL', '"f2" > ''A''']
  - [527, 'c2', false, 'SQL', '"f2" == UPPER("f3") AND NOT "f2" LIKE ''__''']
...
```

## 5.47 box.space._collation

box.space._collation
_collation — это системный спейс со списком видов *сортировки*. Предусмотрены более 270 видов сортировки, и пользователи могут добавлять новые. Один из примеров:

```
tarantool> box.space._collation:select(239)
---
- - [239, 'unicode_uk_s2', 1, 'ICU', 'uk', {'strength': 'secondary'}]
...
```

Пояснения по полям в примере: id = 239, то есть первичный ключ в Tarantool равен 239; name = „unicode_uk_s2“, в соответствии с соглашениями по именованию в Tarantool это сортировка по Юникоду + для региона uk + значение strength 2 уровня; owner = 1, то есть *пользователь уровня admin*; type = „ICU“, то есть применяются правила библиотеки Международных компонентов для Юникода; locale = „uk“, то есть используются настройки для региона Украина; opts = „strength:secondary“, то есть при сортировке учитывается вес первого и второго уровней.

*Представление системного спейса* _collation — _vcollation.

## 5.48 box.space._vcollation

box.space._vcollation
*Представление системного спейса _collation* — _vcollation.

## 5.49 Представления системных спейсов

Представление системного спейса (или „sysview“) — копия системного спейса с доступом только для чтения.

The system space views and the system spaces that they are associated with are: _vcollation, a view of *_collation*, _vfunc, a view of *_func*, _vindex, a view of *_index*, _vpriv, a view of *_priv*, _vsequence, a view of *_sequence*, _vspace, a view of *_space*, _vspace_sequence, a view of *_space_sequence*, _vuser, a view of *_user*.

Структура кортежей в представлении системного спейса идентична структуре кортежам соответствующего спейса. Однако права там обычно отличаются. По умолчанию у обычных пользователей в большинстве системных спейсов нет никаких прав, однако в представлениях системных спейсов у них есть право чтения.

Ситуация по умолчанию: * *У роли „public“* есть право „read“ во всех представлениях системных спейсов, потому что именно так обстоят дела при создании базы данных. * У всех пользователей есть роль „public“, потому что она назначается им автоматически при выполнении *box.schema.user.create()*. * В представлении системного спейса будут содержаться кортежи соответствующего системного спейса, только если у пользователя есть право, связанное с объектом, упомянутом в кортеже. Как результат, если администраторы не изменят права, у остальных пользователей не будет доступа к системному спейсу, но будет доступ к представлению системного спейса, где видны только доступные им объекты.

Например, пользователь „admin“ обычно имеет полный доступ к спейсу _space, поэтому и _vspace для него выглядит как _space. А „guest“ имеет только право на чтение _vspace, так что _vspace для него содержит меньше кортежей, чем _space. Поэтому в большинстве систем пользователю „guest“ чтобы получить список спейсов, следует делать выборку из _vspace.

**Пример**

В этом примере рассмотрим разницу между _vuser и _user. Представление _vuser содержит только кортежи, которые доступны текущему пользователю. Если же у пользователя есть полный набор прав (как у пользователя „admin“), то содержимое _vuser совпадает с содержимым _user.

Чтобы проверить, как работает _vuser, удаленно подключитесь к базе данных Tarantool с помощью net.box и сделайте выборку кортежей из спейса _user в двух случаях: когда у пользователя „guest“ *есть* право на чтение данных из базы и когда его *нет* .

Запустите Tarantool и назначьте права read, write и execute пользователю guest:

```
tarantool> box.cfg{listen = 3301}
---
...
tarantool> box.schema.user.grant('guest', 'read,write,execute', 'universe')
---
...
```

Переключитесь на другой терминал, подключитесь к экземпляру Tarantool и выберите все кортежи из спейса _user:

```
tarantool> conn = require('net.box').connect(3301)
---
...
tarantool> conn.space._user:select{}
---
- - [0, 1, 'guest', 'user', {}]
  - [1, 1, 'admin', 'user', {}]
  - [2, 1, 'public', 'role', {}]
  - [3, 1, 'replication', 'role', {}]
  - [31, 1, 'super', 'role', {}]
...
```

Будут показаны те же пользователи, как если бы вы отправили запрос из своего экземпляра Tarantool как „admin“.

Переключитесь на первый терминал и отзовите право read у пользователя „guest“:

```
tarantool> box.schema.user.revoke('guest', 'read', 'universe')
---
...
```

Переключитесь на другой терминал, остановите сессию (чтобы остановить `tarantool`, нажмите Ctrl+C или Ctrl+D), запустите и подключитесь снова, а затем повторите запрос `conn.space._user:select{}`. Доступ будет запрещен:

```
tarantool> conn.space._user:select{}
---
- error: Read access to space '_user' is denied for user 'guest'
...
```

Однако если вместо этого сделать select из `_vuser`, появятся данные пользователей, доступные пользователю „guest“:

```
tarantool> conn.space._vuser:select{}
---
- - [0, 1, 'guest', 'user', {}]
...
```

## 5.50 box.space._session_settings

`box.space._session_settings`

A temporary system space with settings that affect behavior, particularly SQL behavior, for the current session. It uses a special engine named „service“. Every „service“ tuple is created on the fly, that is, new tuples are made every time `_session_settings` is accessed. Every settings tuple has two fields: `name` (the primary key) and `value`. The tuples“ names and default values are:

- `sql_default_engine`: default *storage engine* for new SQL tables. Default: `memtx`.
- `sql_full_column_names`: use full column names in *SQL result set metadata*. Default: `false`.
- `sql_full_metadata`: whether *SQL result set metadata* includes more than just name and type. Default:`false`.
- `sql_parser_debug`: show parser steps for following statements. Default: `false`.

- `sql_recursive_triggers`: whether a triggered statement can activate a *trigger*. Default: `true`.
- `sql_reverse_unordered_selects`: return result rows in reverse order if there is no *ORDER BY clause*. Default: `false`.
- `sql_select_debug`: show execution steps during *SELECT*. Default:`false`.
- `sql_seq_scan`: allow sequential scans in SQL *SELECT*. Default: `true`.
- `sql_vdbe_debug`: for internal use. Default:`false`.
- `sql_defer_foreign_keys` (removed in *2.11.0*): whether foreign-key checks can wait till commit. Default: `false`.
- `error_marshaling_enabled` (removed in *2.10.0*): whether *error objects* have a special structure. Default: `false`.

Three requests are possible: *select*, *get* and *update*. For example, after `s = box.space._session_settings`, `s:select('sql_default_engine')` probably returns `{'sql_default_engine', 'memtx'}`, and `s:update('sql_default_engine', {{'=', 'value', 'vinyl'}})` changes the default engine to „vinyl“. Updating `sql_parser_debug` or `sql_select_debug` or `sql_vdbe_debug` has no effect unless Tarantool was built with `-DCMAKE_BUILD_TYPE=Debug`. To check if this is so, look at `require('tarantool').build.target`.

ГЛАВА 6

# Администрирование

Tarantool устроен таким образом, что возможно запустить несколько экземпляров программы на одном компьютере.

Здесь мы показываем, как администрировать экземпляры Tarantool с помощью любой из следующих утилит:

- встроенные утилиты `systemd` или
- *tt*, a command-line utility for managing Tarantool-based applications.

---

**Примечание:**

- В отличие от остальной части руководства, в этой главе мы используем общесистемные пути.
- Здесь мы приводим примеры консольного вывода для Fedora.

---

Эта глава включает в себя следующие разделы:

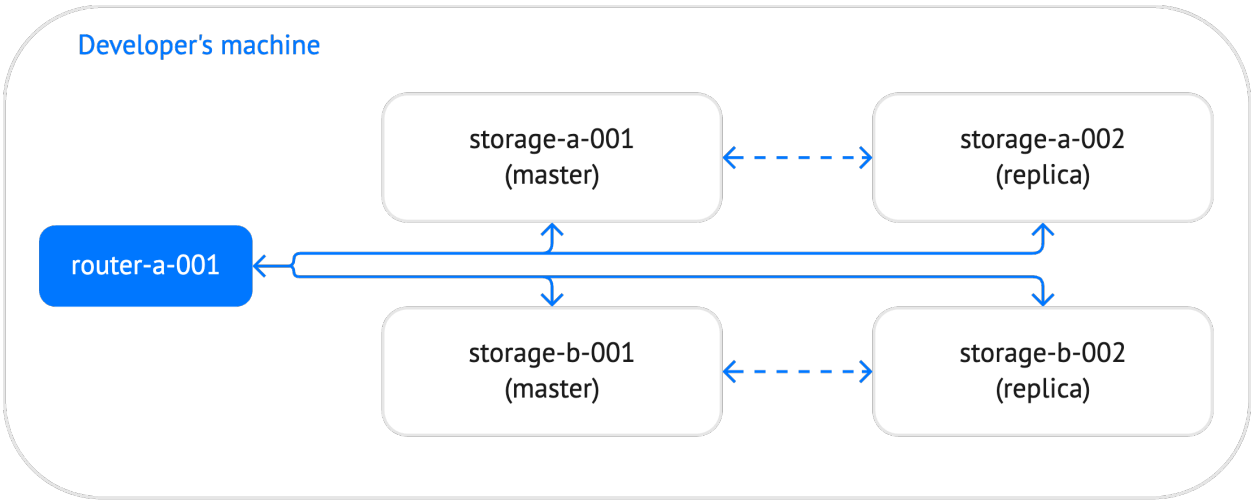
## 6.1 Application environment

This section provides a high-level overview on how to prepare a Tarantool application for deployment and how the application's environment and layout might look. This information is helpful for understanding how to administer Tarantool instances using *tt CLI* in both development and production environments.

The main steps of creating and preparing the application for deployment are:

1. *Initializing a local environment*.
2. *Creating and developing an application*.
3. *Packaging the application*.

In this section, a sharded_cluster application is used as an example. This cluster includes 5 instances: one router and 4 storages, which constitute two replica sets.




### 6.1.1 Initializing a local environment

Before creating an application, you need to set up a local environment for `tt`:

1. Create a home directory for the environment.
2. Run `tt init` in this directory:

```
~/myapp$ tt init
   • Environment config is written to 'tt.yaml'
```

This command creates a default `tt` configuration file `tt.yaml` for a local environment and the directories for applications, control sockets, logs, and other artifacts:

```
~/myapp$ ls
bin  distfiles  include  instances.enabled  modules  templates  tt.yaml
```

Find detailed information about the `tt` configuration parameters and launch modes on the *tt configuration page*.

### 6.1.2 Creating and developing an application

You can create an application in two ways:

- Manually by preparing its layout in a directory inside `instances_enabled`. The directory name is used as the application identifier.
- From a template by using the *tt create* command.

In this example, the application's layout is prepared manually and looks as follows.

```
~/myapp$ tree
.
├── bin
├── distfiles
├── include
├── instances.enabled
│   └── sharded_cluster
```

(continues on next page)

(продолжение с предыдущей страницы)

```
│       ├── config.yaml
│       ├── instances.yaml
│       ├── router.lua
│       ├── sharded_cluster-scm-1.rockspec
│       └── storage.lua
├── modules
├── templates
└── tt.yaml
```

The `sharded_cluster` directory contains the following files:

- `config.yaml`: contains the *configuration* of the cluster. This file might include the entire cluster topology or provide connection settings to a centralized configuration storage.
- `instances.yml`: specifies instances to run in the current environment. For example, on the developer's machine, this file might include all the instances defined in the cluster configuration. In the production environment, this file includes *instances to run on the specific machine*.
- `router.lua`: includes code specific for a *router*.
- `sharded_cluster-scm-1.rockspec`: specifies the required external dependencies (for example, `vshard`).
- `storage.lua`: includes code specific for *storages*.

You can find the full example here: sharded_cluster.

### 6.1.3 Packaging the application

To package the ready application, use the *tt pack* command. This command can create an installable DEB/RPM package or generate `.tgz` archive.

The structure below reflects the content of the packed `.tgz` archive for the sharded_cluster application:

```
~/myapp$ tree -a
.
├── bin
│   ├── tarantool
│   └── tt
├── include
├── instances.enabled
│   └── sharded_cluster -> ../sharded_cluster
├── modules
├── sharded_cluster
│   ├── .rocks
│   │   └── share
│   │       └── ...
│   ├── config.yaml
│   ├── instances.yaml
│   ├── router.lua
│   ├── sharded_cluster-scm-1.rockspec
│   └── storage.lua
└── tt.yaml
```

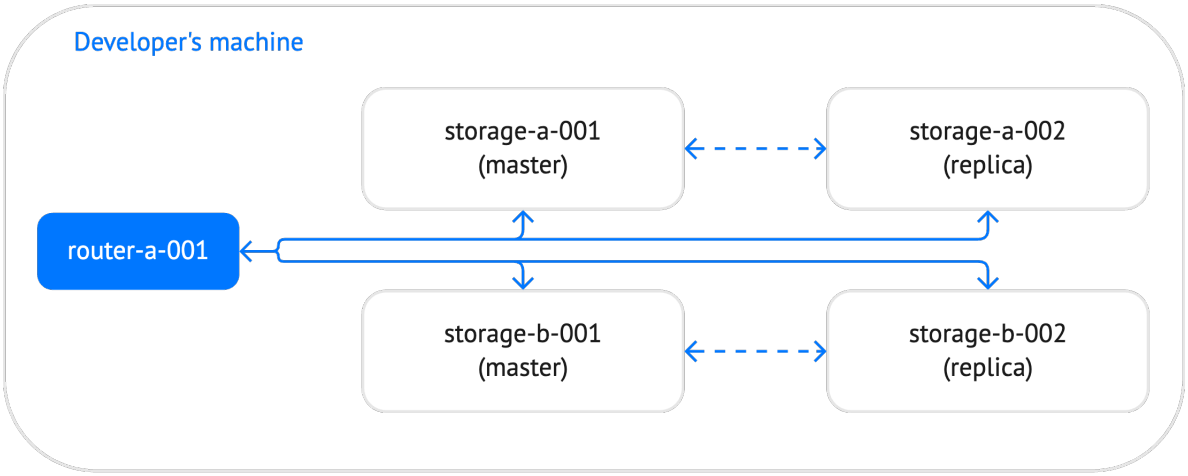
The application's layout looks similar to the one defined when *developing the application* with some differences:

- `bin`: contains the `tarantool` and `tt` binaries packed with the application bundle.
- `instances.enabled`: contains a symlink to the packed `sharded_cluster` application.
- `sharded_cluster`: a packed application. In addition to files created during the application development, includes the `.rocks` directory containing application dependencies (for example, `vshard`).
- `tt.yaml`: a `tt` configuration file.

### Instances to run

One more difference for a deployed application is the content of the `instances.yaml` file that specifies instances to run in the current environment.

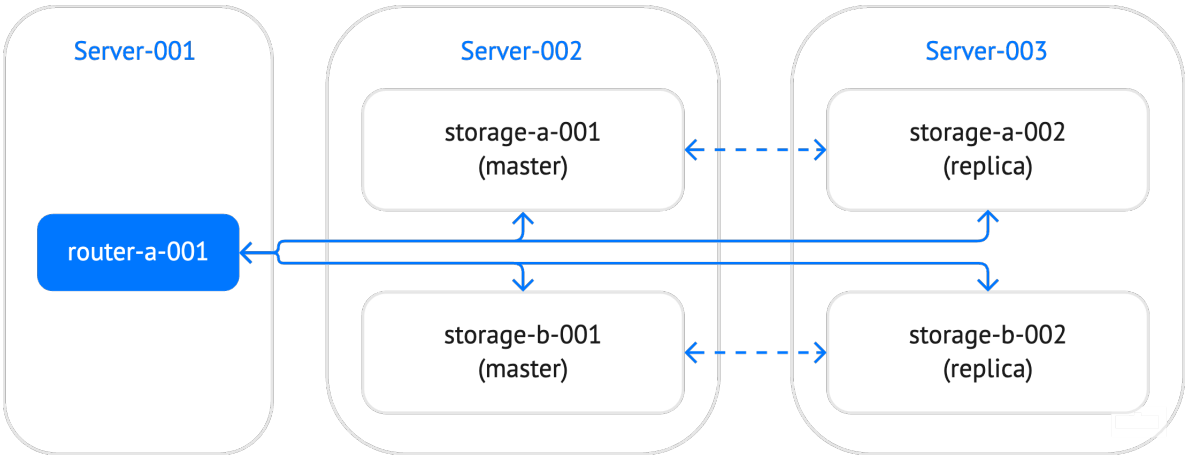
- On the developer's machine, this file might include all the instances defined in the cluster configuration.




`instances.yaml`:

```
storage-a-001:
storage-a-002:
storage-b-001:
storage-b-002:
router-a-001:
```

- In the production environment, this file includes instances to run on the specific machine.



`instances.yaml` (Server-001):

```
router-a-001:
```

`instances.yaml` (Server-002):

```
storage-a-001:
storage-b-001:
```

`instances.yaml` (Server-003):

```
storage-a-002:
storage-b-002:
```

The *Starting and stopping instances* section describes how to start and stop Tarantool instances.

## 6.2 Starting and stopping instances

This section describes how to manage instances in a Tarantool cluster using the *tt* utility. A cluster can include multiple instances that run different code. A typical example is a cluster application that includes router and storage instances. Particularly, you can perform the following actions:

- start all instances in a cluster or only specific ones
- check the status of instances
- connect to a specific instance
- stop all instances or only specific ones

To get more context on how the application's environment might look, refer to *Application environment*.

---

**Примечание:** In this section, a sharded_cluster application is used to demonstrate how to start, stop, and manage instances in a cluster.

---

### 6.2.1 Starting Tarantool instances

#### Starting instances using the tt utility

The *tt* utility is the recommended way to start Tarantool instances.

```
$ tt start sharded_cluster
   • Starting an instance [sharded_cluster:storage-a-001]...
   • Starting an instance [sharded_cluster:storage-a-002]...
   • Starting an instance [sharded_cluster:storage-b-001]...
   • Starting an instance [sharded_cluster:storage-b-002]...
   • Starting an instance [sharded_cluster:router-a-001]...
```

After the cluster has started and worked for some time, you can find its artifacts in the directories specified in the `tt` configuration. These are the default locations in the local *launch mode*:

- `sharded_cluster/var/log/<instance_name>/` – instance *logs*.
- `sharded_cluster/var/lib/<instance_name>/` – *snapshots and write-ahead logs*.

- `sharded_cluster/var/run/<instance_name>/` – control sockets and PID files.

In the system launch mode, artifacts are created in these locations:

- `/var/log/tarantool/<instance_name>/`
- `/var/lib/tarantool/<instance_name>/`
- `/var/run/tarantool/<instance_name>/`

#### Starting an instance using the tarantool command

The `tarantool` command provides additional *options* that might be helpful for development purposes. Below is the syntax for starting a Tarantool instance configured in a file:

```
$ tarantool [OPTION ...] --name INSTANCE_NAME --config CONFIG_FILE_PATH
```

The command below starts `router-a-001` configured in the `config.yaml` file:

```
$ tarantool --name router-a-001 --config config.yaml
```

### 6.2.2 Basic instance management

Most of the commands described in this section can be called with or without an instance name. Without the instance name, they are executed for all instances defined in `instances.yaml`.

#### Checking an instance's status

To check the status of instances, execute *tt status*:

```
$ tt status sharded_cluster
INSTANCE                          STATUS      PID
sharded_cluster:storage-a-001     RUNNING     2023
sharded_cluster:storage-a-002     RUNNING     2026
sharded_cluster:storage-b-001     RUNNING     2020
sharded_cluster:storage-b-002     RUNNING     2021
sharded_cluster:router-a-001      RUNNING     2022
```

To check the status of a specific instance, you need to specify its name:

```
$ tt status sharded_cluster:storage-a-001
INSTANCE                          STATUS      PID
sharded_cluster:storage-a-001     RUNNING     2023
```

### Connecting to an instance

To connect to the instance, use the *tt connect* command:

```
$ tt connect sharded_cluster:storage-a-001
   • Connecting to the instance...
   • Connected to sharded_cluster:storage-a-001

sharded_cluster:storage-a-001>
```

In the instance's console, you can execute commands provided by the *box* module. For example, *box.info* can be used to get various information about a running instance:

```
sharded_cluster:storage-a-001> box.info.ro
---
- false
...
```

### Restarting instances

To restart an instance, use *tt restart*:

```
$ tt restart sharded_cluster:storage-a-002
```

After executing `tt restart`, you need to confirm this operation:

```
Confirm restart of 'sharded_cluster:storage-a-002' [y/n]: y
   • The Instance sharded_cluster:storage-a-002 (PID = 2026) has been terminated.
   • Starting an instance [sharded_cluster:storage-a-002]...
```

### Stopping instances

To stop the specific instance, use *tt stop* as follows:

```
$ tt stop sharded_cluster:storage-a-002
```

You can also stop all the instances at once as follows:

```
$ tt stop sharded_cluster
   • The Instance sharded_cluster:storage-b-001 (PID = 2020) has been terminated.
   • The Instance sharded_cluster:storage-b-002 (PID = 2021) has been terminated.
   • The Instance sharded_cluster:router-a-001 (PID = 2022) has been terminated.
   • The Instance sharded_cluster:storage-a-001 (PID = 2023) has been terminated.
   • can't "stat" the PID file. Error: "stat /home/testuser/myapp/instances.enabled/
↪sharded_cluster/var/run/storage-a-002/tt.pid: no such file or directory"
```

**Примечание:** The error message indicates that `storage-a-002` is already not running.

#### Removing instance artifacts

The *tt clean* command removes instance artifacts (such as logs or snapshots):

```
$ tt clean sharded_cluster
   • List of files to delete:

   • /home/testuser/myapp/instances.enabled/sharded_cluster/var/log/storage-a-001/tt.log
   • /home/testuser/myapp/instances.enabled/sharded_cluster/var/lib/storage-a-001/
↪00000000000000001062.snap
   • /home/testuser/myapp/instances.enabled/sharded_cluster/var/lib/storage-a-001/
↪00000000000000001062.xlog
   • ...

Confirm [y/n]:
```

Enter `y` and press `Enter` to confirm removing of artifacts for each instance.

---

**Примечание:** The `-f` option of the `tt clean` command can be used to remove the files without confirmation.

---

### 6.2.3 Preloading Lua scripts and modules

Tarantool supports loading and running chunks of Lua code before starting instances. To load or run Lua code immediately upon Tarantool startup, specify the `TT_PRELOAD` environment variable. Its value can be either a path to a Lua script or a Lua module name:

- To run the Lua script `preload_script.lua` from the `sharded_cluster` directory, set `TT_PRELOAD` as follows:

  ```
  $ TT_PRELOAD=preload_script.lua tt start sharded_cluster
  ```

  Tarantool runs the `preload_script.lua` code, waits for it to complete, and then starts instances.

- To load the `preload_module` from the `sharded_cluster` directory, set `TT_PRELOAD` as follows:

  ```
  $ TT_PRELOAD=preload_module tt start sharded_cluster
  ```

  ---

  **Примечание:** `TT_PRELOAD` values that end with `.lua` are considered scripts, so avoid module names with this ending.

  ---

To load several scripts or modules, pass them in a single quoted string, separated by semicolons:

```
$ TT_PRELOAD="preload_script.lua;preload_module" tt start sharded_cluster
```

If an error happens during the execution of the preload script or module, Tarantool reports the problem and exits.

### 6.2.4 tarantool command-line options

Options that can be passed when *starting a Tarantool instance*:

`-h, --help`
: Print an annotated list of all available options and exit.

`--help-env-list`
: **Since:** *3.0.0*.

  Show a list of *environment variables* that can be used to configure Tarantool.

`-v, -V, --version`
: Print the product name and version.

  **Example**

```
$ tarantool --version
Tarantool Enterprise 3.0.0-beta1-2-gcbb569b4c-r607-gc64
Target: Linux-x86_64-RelWithDebInfo
...
```

  In this example:

  - `3.0.0` is a Tarantool version. Tarantool follows semantic versioning, which is described in the *Tarantool release policy* section.
  - `Target` is the platform Tarantool is built on. Platform-specific details may follow this line.

`-c, --config PATH`
: **Since:** *3.0.0*.

  Set a path to a *YAML configuration file*. You can also configure this value using the `TT_CONFIG` environment variable.

  See also: *Starting an instance using the tarantool command*

`-n, --name INSTANCE`
: **Since:** *3.0.0*.

  Set the name of an instance to run. You can also configure this value using the `TT_INSTANCE_NAME` environment variable.

  See also: *Starting an instance using the tarantool command*

`-i`
: Enter an *interactive mode*.

  **Example**

```
$ tarantool -i
```

`-e EXPR`
: Execute the „EXPR“ string. See also: lua man page.

  **Example**

```
$ tarantool -e 'print("Hello, world!")'
Hello, world!
```

`-l NAME`
: Require the „NAME“ library. See also: lua man page.

**Example**

```
$ tarantool -l luatest.coverage script.lua
```

`-j cmd`
: Perform a LuaJIT control command. See also: Command Line Options.

**Example**

```
$ tarantool -j off app.lua
```

`-b ...`
: Save or list bytecode. See also: Command Line Options.

**Example**

```
$ tarantool -b test.lua test.out
```

`-d SCRIPT`
: Activate a debugging session for „SCRIPT“. See also: luadebug.lua.

**Example**

```
$ tarantool -d app.lua
```

`--`
: Stop handling options. See also: lua man page.

`-`
: Stop handling options and execute the standard input as a file. See also: lua man page.

## 6.3 Managing modules

This section covers the installation and reloading of Tarantool modules. To learn about writing your own module and contributing it, check the *Contributing a module* section.

### 6.3.1 Установка модуля

Модули на Lua и C от разработчиков Tarantool и сторонних разработчиков доступны здесь:

- Репозиторий модулей Tarantool (см. *ниже*)
- Репозитории deb/rpm Tarantool (см. *ниже*)

#### Установка модуля из репозитория

Для получения подробной информации см. README в репозитории tarantool/rocks.

### Установка модуля из deb/rpm

Выполните следующие действия:

1. Установите Tarantool в соответствии с рекомендациями на странице загрузки.
2. Install the module you need. Look up the module's name on Tarantool rocks page and put the prefix «tarantool-» before the module name to avoid ambiguity:

```
$ # для Ubuntu/Debian:
$ sudo apt-get install tarantool-<module-name>

$ # для RHEL/CentOS/Amazon:
$ sudo yum install tarantool-<module-name>
```

   Например, чтобы установить модуль vshard на Ubuntu, введите:

```
$ sudo apt-get install tarantool-vshard
```

Теперь можно:

- загружать любой модуль с помощью

```
tarantool> name = require('module-name')
```

  например:

```
tarantool> vshard = require('vshard')
```

- локально находить установленные модули с помощью `package.path` (Lua) или `package.cpath` (C):

```
tarantool> package.path
---
- ./?.lua;./?/init.lua; /usr/local/share/tarantool/?.lua;/usr/local/share/
tarantool/?/init.lua;/usr/share/tarantool/?.lua;/usr/share/tarantool/?/ini
t.lua;/usr/local/share/lua/5.1/?.lua;/usr/local/share/lua/5.1/?/init.lua;/
usr/share/lua/5.1/?.lua;/usr/share/lua/5.1/?/init.lua;
...

tarantool> package.cpath
---
- ./?.so;/usr/local/lib/x86_64-linux-gnu/tarantool/?.so;/usr/lib/x86_64-li
nux-gnu/tarantool/?.so;/usr/local/lib/tarantool/?.so;/usr/local/lib/x86_64
-linux-gnu/lua/5.1/?.so;/usr/lib/x86_64-linux-gnu/lua/5.1/?.so;/usr/local/
lib/lua/5.1/?.so;
...
```

---

**Примечание:** Знаки вопроса стоят вместо имени модуля, которое было указано ранее при вызове `require('module-name')`.

---

### 6.3.2 Перезагрузка модуля

Любое приложение или модуль Tarantool можно перезагрузить с нулевым временем простоя.

#### Перезагрузка модуля на Lua

Ниже представлен пример, который иллюстрирует наиболее типичный случай – «обновление и перезагрузка».

---

**Примечание:** In this example, we use recommended *administration practices* based on *instance files* and *tt* utility.

---

1. Обновите файлы приложения.

   Например, модуль в `/usr/share/tarantool/app.lua`:

```
local function start()
  -- начальная версия
  box.once("myapp:v1.0", function()
    box.schema.space.create("somedata")
    box.space.somedata:create_index("primary")
    ...
  end)

  -- код миграции с 1.0 на 1.1
  box.once("myapp:v1.1", function()
    box.space.somedata.index.primary:alter(...)
    ...
  end)

  -- код миграции с 1.1 на 1.2
  box.once("myapp:v1.2", function()
    box.space.somedata.index.primary:alter(...)
    box.space.somedata:insert(...)
    ...
  end)
end

-- запустить файберы в фоновом режиме, если необходимо

local function stop()
  -- остановить все файберы, работающие в фоновом режиме, и очистить ресурсы
end

local function api_for_call(xxx)
  -- сделать что-то
end

return {
  start = start,
  stop = stop,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  api_for_call = api_for_call
}
```

2. Обновить *файл экземпляра*.

   Например, `/etc/tarantool/instances.enabled/my_app.lua`:

```
#!/usr/bin/env tarantool
--
-- пример горячей перезагрузки кода
--

box.cfg({listen = 3302})

-- ВНИМАНИЕ: правильно выполните разгрузку!
local app = package.loaded['app']
if app ~= nil then
  -- остановите старую версию приложения
  app.stop()
  -- разгрузите приложение
  package.loaded['app'] = nil
  -- разгрузите все зависимости
  package.loaded['somedep'] = nil
end

-- загрузите приложение
log.info('require app')
app = require('app')

-- запустите приложение
app.start({some app options controlled by sysadmins})
```

   Самое главное – правильно разгрузить приложение и его зависимости.

3. Вручную перезагрузите файл приложения.

   For example, using `tt`:

```
$ tt connect my_app -f /etc/tarantool/instances.enabled/my_app.lua
```

### Перезагрузка модуля на C

После компиляции новой версии модуля на C (библиотека общего пользования `*.so`), вызовите функцию *box.schema.func.reload(„module-name“)* из Lua-скрипта для перезагрузки модуля.

## 6.4 Журналирование

Each Tarantool instance logs important events to its own log file. For instances started with *tt*, the log location is defined by the `log_dir` parameter in the *tt configuration*. By default, it's `/var/log/tarantool` in the `tt` *system mode*, and the `var/log` subdirectory of the `tt` working directory in the *local mode*. In the specified location, `tt` creates separate directories for each instance's logs.

To check how logging works, write something to the log using the *log* module:

```
$ tt connect application
   • Connecting to the instance...
   • Connected to application

application> require('log').info("Hello for the manual readers")
---
...
```

Затем проверим содержимое журнала:

```
$ tail instances.enabled/application/var/log/instance001/tt.log
2024-04-09 17:34:29.489 [49502] main/106/gc I> wal/engine cleanup is resumed
2024-04-09 17:34:29.489 [49502] main/104/interactive/box.load_cfg I> set 'instance_name'␣
↪configuration option to "instance001"
2024-04-09 17:34:29.489 [49502] main/104/interactive/box.load_cfg I> set 'custom_proc_
↪title' configuration option to "tarantool - instance001"
2024-04-09 17:34:29.489 [49502] main/104/interactive/box.load_cfg I> set 'log_nonblock'␣
↪configuration option to false
2024-04-09 17:34:29.489 [49502] main/104/interactive/box.load_cfg I> set 'replicaset_name
↪' configuration option to "replicaset001"
2024-04-09 17:34:29.489 [49502] main/104/interactive/box.load_cfg I> set 'listen'␣
↪configuration option to [{"uri":"127.0.0.1:3301"}]
2024-04-09 17:34:29.489 [49502] main/107/checkpoint_daemon I> scheduled next checkpoint␣
↪for Tue Apr  9 19:08:04 2024
2024-04-09 17:34:29.489 [49502] main/104/interactive/box.load_cfg I> set 'metrics'␣
↪configuration option to {"labels":{"alias":"instance001"},"include":["all"],"exclude
↪":[]}
2024-04-09 17:34:29.489 [49502] main I> entering the event loop
2024-04-09 17:34:38.905 [49502] main/116/console/unix/:/tarantool I> Hello for the␣
↪manual readers
```

### 6.4.1 Log rotation

When *logging to a file*, the system administrator must ensure logs are rotated timely and do not take up all the available disk space. The recommended way to prevent log files from growing infinitely is using an external log rotation program, for example, `logrotate`, which is pre-installed on most mainstream Linux distributions.

A Tarantool log rotation configuration for `logrotate` can look like this:

```
# /var/log/tarantool/<env>/<app>/<instance>/*.log
/var/log/tarantool/*/*/*/*.log {
    daily
    size 512k
    missingok
    rotate 10
    compress
    delaycompress
    sharedscripts # Run tt logrotate only once after all logs are rotated.
    postrotate
        /usr/bin/tt -S logrotate
    endscript
}
```

In this configuration, *tt logrotate* is called after each log rotation to reopen the instance log files after they are moved by the `logrotate` program.

There is also the built-in function *log.rotate()*, which you can call on an instance to reopen its log file after rotation.

To learn about log rotation in the deprecated `tarantoolctl` utility, check its *documentation*.

### 6.4.2 Log destination

Tarantool can write its logs to a log file, to `syslog`, or to a specified program through a pipe. For example, to send logs to `syslog`, specify the *log.to* parameter as follows:

```
log:
  to: syslog
  syslog:
    server: '127.0.0.1:514'
```

## 6.5 Безопасность

Tarantool разрешает два типа подключений:

- With *console.listen()* function from `console` module, you can set up a port which can be used to open an administrative console to the server. This is for administrators to connect to a running instance and make requests. `tt` invokes `console.listen()` to create a control socket for each started instance.
- Используя параметр *box.cfg{listen=...}* из модуля `box`, можно настроить бинарный порт для соединений, которые читают и пишут в базу данных или вызывают хранимые процедуры.

Если вы подключены к административной консоли:

- Клиент-серверный протокол – это простой текст.

- Пароль не требуется.
- Пользователь автоматически получает права администратора.
- Каждая команда напрямую обрабатывается встроенным интерпретатором Lua.

Поэтому порты для административной консоли следует настраивать очень осторожно. Если это TCP-порт, он должен быть открыть только для определенного IP-адреса. В идеале вместо TCP-порта лучше настроить доменный Unix-сокет, который требует наличие прав доступа к серверной машине. Тогда типичная настройка порта для административной консоли будет выглядеть следующим образом:

```
console.listen('/var/lib/tarantool/socket_name.sock')
```

а типичный *URI* для соединения будет таким:

```
/var/lib/tarantool/socket_name.sock
```

if the listener has the privilege to write on `/var/lib/tarantool` and the connector has the privilege to read on `/var/lib/tarantool`. Alternatively, to connect to an admin console of an instance started with `tt`, use *tt connect*.

Выяснить, является ли некоторый TCP-порт портом для административной консоли, можно с помощью `telnet`. Например:

```
$ telnet 0 3303
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
Tarantool 2.1.0 (Lua console)
type 'help' for interactive help
```

В этом примере в ответе от сервера нет слова «binary» и есть слова «Lua console». Это значит, что мы успешно подключились к порту для административной консоли и можем вводить администраторские запросы на этом терминале.

Если вы подключены к бинарному порту:

- Клиент-серверный протокол – *бинарный*.
- Автоматически выбирается пользователь „*guest*“.
- Для смены пользователя необходимо пройти аутентификацию.

For ease of use, `tt connect` command automatically detects the type of connection during handshake and uses *EVAL* binary protocol command when it's necessary to execute Lua commands over a binary connection. To execute EVAL, the authenticated user must have global «EXECUTE» privilege.

Поэтому при невозможности подключиться к машине по `ssh` системный администратор может получить **удаленный** доступ к экземпляру, создав пользователя Tarantool с глобальными «EXECUTE»-правами и непустым паролем.

## 6.6 Управление доступом

Tarantool enables flexible management of access to various database resources. The main concepts of Tarantool access control system are as follows:

- A *user* is a person or program that interacts with a Tarantool instance.
- An *object* is an entity to which access can be granted, for example, a space, an index, or a function.
- A *privilege* allows a user to perform certain operations on specific objects, for example, creating spaces, reading or updating data.
- A *role* is a named collection of privileges that can be granted to a user.

### 6.6.1 Overview

#### Пользователи

A user identifies a person or program that interacts with a Tarantool instance. There might be different types of users, for example:

- A database administrator responsible for the overall management and administration of a database. An administrator can create other users and grant them specified privileges.
- A user with limited access to certain data and stored functions. Such users can get their privileges from the database administrator.
- Users used in communications between Tarantool instances. For example, such users can be created to maintain replication and sharding in a Tarantool cluster.

There are two built-in users in Tarantool:

- `admin` is a user with all available administrative privileges. If the connection uses an *admin-console port*, the current user is `admin`. For example, `admin` is used when connecting to an instance using *tt connect* locally using the instance name:

  ```
  $ tt connect app:instance001
  ```

  To allow remote *binary port* connections using the `admin` user, you need to *set a password*.

- `guest` is a user with minimum privileges used by default for remote *binary port* connections. For example, `guest` is used when connecting to an instance using *tt connect* using the IP address and port without specifying the name of a user:

  ```
  $ tt connect 192.168.10.10:3301
  ```

  **Предупреждение:** Given that the `guest` user allows unauthenticated access to Tarantool instances, it is not recommended to grant additional privileges to this user. For example, granting the `execute` access to *universe* allows *remote code execution* on instances.

**Примечание:** Information about users is stored in the *_user* space.

### Пароли

Any user (except `guest`) may have a password. If a password is not set, a user cannot connect to Tarantool instances.

Tarantool *password hashes* are stored in the *_user* system space. By default, Tarantool uses the `CHAP` protocol to authenticate users and applies `SHA-1` hashing to *passwords*. So, if the password is „123456“, the stored hash is a string like „a7SDfrdDKRBe5FaN2n3GftLKKtk=“. In the Enterprise Edition, you can enable `PAP` *authentication* with the `SHA256` hashing algorithm.

Tarantool Enterprise Edition allows you to improve database security by enforcing the use of strong passwords, setting up a maximum password age, and so on. Learn more from the *Authentication* topic.

### Objects

An object is a securable entity to which access can be granted. Tarantool has a number of objects that enable flexible management of access to data, stored functions, specific actions, and so on.

Below are a few examples of objects:

- `universe` represents a database (*box.schema*) that contains database objects, including spaces, indexes, users, roles, sequences, and functions. Granting privileges to `universe` gives a user access to any object in a database.
- `space` enables granting privileges to user-created or system *spaces*.
- `function` enables granting privileges to *functions*.

---

**Примечание:** The full list of object types is available in the *Object types* section.

---

### Privileges

The privileges granted to a user determine which operations the user can perform, for example:

- The `read` and `write` permissions granted to the `space` *object* allow a user to read or modify data in the specified space.
- The `create` permission granted to the `space` object allows a user to create new spaces.
- The `execute` permission granted to the `function` object allows a user to execute the specified function.
- The `session` permission granted to a user allows connecting to an instance over IPROTO.

Note that some privileges might require read and write access to certain system spaces. For example, the `create` permission granted to the `space` object requires `read` and `write` permissions to the *_space* system space. Similarly, granting the ability to create functions requires `read` and `write` access to the *_func* space.

---

**Примечание:** Information about privileges is stored in the *_priv* space.

---

### Роли

A role is a container for *privileges* that can be granted to users. Roles can also be assigned to other roles, creating a role hierarchy.

There are the following built-in roles in Tarantool:

- `super` has all available administrative permissions.
- `public` has certain read permissions. This role is automatically granted to new users when they are created.
- `replication` can be granted to a user used to maintain *replication* in a cluster.
- `sharding` can be granted to a user used to maintain *sharding* in a cluster.

---

**Примечание:** The `sharding` role is created only if an instance is managed using *YAML configuration*.

---

Below are a few diagrams that demonstrate how privileges can be granted to a user without and with using roles.

- In this example, a user gets privileges directly without using roles.

```
user1 ── privilege1
    ├─── privilege2
    └─── privilege3
```

- In this example, a user gets all privileges provided by `role1` and specific privileges assigned directly.

```
user1 ── role1 ── privilege1
    │         └─── privilege2
    ├─── privilege3
    └─── privilege4
```

- In this example, `role2` is granted to `role1`. This means that a user with `role1` subsequently gets all privileges from both roles `role1` and `role2`.

```
user1 ── role1 ── privilege1
    │         ├─── privilege2
    │         └─── role2
    │                   ├─── privilege3
    │                   └─── privilege4
    ├─── privilege5
    └─── privilege6
```

---

**Примечание:** Information about roles is stored in the *_user* space.

---

#### Object owners

An owner of a database *object* is the user who created it. The owner of the database and the owner of objects that are created initially (the system spaces and the default users) is the `admin` *user*.

Owners automatically have *privileges* for objects they create. They can *share these privileges* with other users or roles using `box.schema.user.grant()` and `box.schema.role.grant()`.

---

**Примечание:** Information about users who gave the specified privileges is stored in the *_priv* space.

---

#### Sessions

A session is the state of a connection to Tarantool. The session contains:

- An integer ID identifying the connection.
- The current *user* associated with the connection.
- The text description of the connected peer.
- A session's local state, such as Lua variables and functions.

In Tarantool, a single session can execute multiple concurrent transactions. Each transaction is identified by a unique integer ID, which can be queried at the start of the transaction using *box.session.sync()*.

---

**Примечание:** Чтобы отследить все подключения и отключения, можно использовать *триггеры соединений и аутентификации*.

---

### 6.6.2 Managing users

#### Creating a user

To create a new user, call *box.schema.user.create()*. In the example below, a user is created without a password:

```
box.schema.user.create('testuser')
```

In this example, the password is specified in the `options` parameter:

```
box.schema.user.create('testuser', { password = 'foobar' })
```

#### Changing passwords

To set or change a user's password, use *box.schema.user.passwd()*. In the example below, a user password is set for a currently logged-in user:

```
box.schema.user.passwd('foobar')
```

To set the password for the specified user, pass a username and password as shown below:

```
box.schema.user.passwd('testuser', 'foobar')
```

**Примечание:** *box.schema.user.password()* returns a hash of the specified password.

### Granting privileges to a user

To grant the specified privileges to a user, use the *box.schema.user.grant()* function. In the example below, `testuser` gets read permissions to the `writers` space and read/write permissions to the `books` space:

```
box.schema.user.grant('testuser', 'read', 'space', 'writers')
box.schema.user.grant('testuser', 'read,write', 'space', 'books')
```

Learn more about granting privileges to different types of objects from *Granting privileges*.

### Getting a user's information

To check whether the specified user exists, call *box.schema.user.exists()*:

```
box.schema.user.exists('testuser')
--[[
- true
--]]
```

To get information about privileges granted to a user, call *box.schema.user.info()*:

```
box.schema.user.info('testuser')
--[[
- - - execute
    - role
    - public
  - - read
    - space
    - writers
  - - read,write
    - space
    - books
  - - session,usage
    - universe
    -
  - - alter
    - user
    - testuser
--]]
```

In the example above, `testuser` has the following privileges:

- The `execute` permission to the `public` role means that this role is *assigned to the user*.
- The `read` permission to the `writers` space means that the user can read data from this space.
- The `read,write` permissions to the `books` space mean that the user can read and modify data in this space.
- The `session,usage` permissions to `universe` mean the following:
  - `session`: the user can authenticate over an IPROTO connection.

  - `usage`: lets the user use their privileges on database objects (for example, read and modify data in a space).

- The `alter` permission lets `testuser` modify its own settings, for example, a password.

### Revoking user's privileges

To revoke the specified privileges, use the *box.schema.user.revoke()* function. In the example below, write access to the `books` space is revoked:

```
box.schema.user.revoke('testuser', 'write', 'space', 'books')
```

Revoking the `session` permission to `universe` can be used to disallow a user to connect to a Tarantool instance:

```
box.schema.user.revoke('testuser', 'session', 'universe')
```

### Changing the current user

The current user name can be found using *box.session.user()*.

```
box.session.user()
--[[
- admin
--]]
```

Текущего пользователя можно изменить:

- For an *admin-console* connection: using *box.session.su()*:

  ```
  box.session.su('testuser')
  box.session.user()
  --[[
  - testuser
  --]]
  ```

- For a binary port connection: using the *AUTH protocol command*, supported by most clients.
- For a binary-port connection invoking a stored function with the CALL command: if the *SETUID* property is enabled for the function, Tarantool temporarily replaces the current user with the function's creator, with all the creator's privileges, during function execution.

### Dropping users

To drop the specified user, call *box.schema.user.drop()*:

```
box.schema.user.drop('testuser')
```

### 6.6.3 Managing roles

#### Creating a role

To create a new role, call *box.schema.role.create()*. In the example below, two roles are created:

```
box.schema.role.create('books_space_manager')
box.schema.role.create('writers_space_reader')
```

#### Granting privileges to a role

To grant the specified privileges to a role, use the *box.schema.role.grant()* function. In the example below, the `books_space_manager` role gets read and write permissions to the `books` space:

```
box.schema.role.grant('books_space_manager', 'read,write', 'space', 'books')
```

The `writers_space_reader` role gets read permissions to the `writers` space:

```
box.schema.role.grant('writers_space_reader', 'read', 'space', 'writers')
```

Learn more about granting privileges to different types of objects from *Granting privileges*.

---

**Примечание:** Not all privileges can be granted to roles. Learn more from *Permissions*.

---

#### Granting a role to a role

Roles can be assigned to other roles. In the example below, the newly created `all_spaces_manager` role gets all privileges granted to `books_space_manager` and `writers_space_reader`:

```
box.schema.role.create('all_spaces_manager')
box.schema.role.grant('all_spaces_manager', 'books_space_manager')
box.schema.role.grant('all_spaces_manager', 'writers_space_reader')
```

#### Granting a role to a user

To grant the specified role to a *user*, use the `box.schema.user.grant()` function. In the example below, `testuser` gets privileges granted to the `books_space_manager` and `writers_space_reader` roles:

```
box.schema.user.grant('testuser', 'books_space_manager')
box.schema.user.grant('testuser', 'writers_space_reader')
```

### Getting a role's information

To check whether the specified role exists, call *box.schema.role.exists()*:

```
box.schema.role.exists('books_space_manager')
--[[
- true
--]]
```

To get information about privileges granted to a role, call *box.schema.role.info()*:

```
box.schema.role.info('books_space_manager')
--[[
- - - read,write
    - space
    - books
--]]
```

If a role has the `execute` permission to other roles, this means that these roles are *granted to this parent role*:

```
box.schema.role.info('all_spaces_manager')
--[[
- - - execute
    - role
    - books_space_manager
  - - execute
    - role
    - writers_space_reader
--]]
```

### Revoking a role from a user

To revoke the specified role from a user, revoke the `execute` privilege for this role using the *box.schema.user.revoke()* function. In the example below, the `books_space_reader` role is revoked from `testuser`:

```
box.schema.user.revoke('testuser', 'execute', 'role', 'writers_space_reader')
```

To revoke role's privileges, use *box.schema.role.revoke()*.

### Dropping roles

To drop the specified role, call *box.schema.role.drop()*:

```
box.schema.role.drop('writers_space_reader')
```

### 6.6.4 Granting privileges

To grant the specified privileges to a user or role, use the *box.schema.user.grant()* and *box.schema.role.grant()* functions, which have similar signatures and accept the same set of arguments. For example, the `box.schema.user.grant()` signature looks as follows:

```
box.schema.user.grant(username, permissions, object-type, object-name[, {options}])
```

- `username`: the name of the user that gets the specified privileges.
- `permissions`: a string value that represents *permissions* granted to the user. If there are several permissions, they should be separated by commas without a space.
- `object-type`: a type of *object* to which permissions are granted.
- `object-name`: the name of the object to which permissions are granted. An empty string (`""`) or `nil` provided instead of `object-name` grants the specified permissions to all objects of the specified type.

  ---

  **Примечание:** `object-name` is ignored for the following combinations of permissions and object types:

  – Any permission granted to `universe`.

  – The `create` and `drop` permissions for the following object types: `user`, `role`, `space`, `function`, `sequence`.

  – The `execute` permission for the following object types: `lua_eval`, `lua_call`, `sql`.

  ---

#### Any object

In the example below, `testuser` gets privileges allowing them to create any object of any type:

```
box.schema.user.grant('testuser','read,write,create','universe')
```

In this example, `testuser` can grant access to objects that `testuser` created:

```
box.schema.user.grant('testuser','write','space','_priv')
```

#### Spaces

##### Creating and altering spaces

In the example below, `testuser` gets privileges allowing them to create *spaces*:

```
box.schema.user.grant('testuser','create','space')
box.schema.user.grant('testuser','write', 'space', '_schema')
box.schema.user.grant('testuser','write', 'space', '_space')
```

As you can see, the ability to create spaces also requires `write` access to certain system spaces.

To allow `testuser` to drop a space that has associated objects, add the following privileges:

```
box.schema.user.grant('testuser','create,drop','space')
box.schema.user.grant('testuser','write','space','_schema')
box.schema.user.grant('testuser','write','space','_space')
box.schema.user.grant('testuser','write','space','_space_sequence')
box.schema.user.grant('testuser','read','space','_trigger')
box.schema.user.grant('testuser','read','space','_fk_constraint')
box.schema.user.grant('testuser','read','space','_ck_constraint')
box.schema.user.grant('testuser','read','space','_func_index')
```

### Creating and altering indexes

In the example below, `testuser` gets privileges allowing them to create *indexes* in the „writers“ space:

```
box.schema.user.grant('testuser','create,read','space','writers')
box.schema.user.grant('testuser','read,write','space','_space_sequence')
box.schema.user.grant('testuser','write', 'space', '_index')
```

To allow `testuser` to alter indexes in the `writers` space, grant the privileges below. This example assumes that indexes in the `writers` space are not created by `testuser`.

```
box.schema.user.grant('testuser','alter','space','writers')
box.schema.user.grant('testuser','read','space','_space')
box.schema.user.grant('testuser','read','space','_index')
box.schema.user.grant('testuser','read','space','_space_sequence')
box.schema.user.grant('testuser','write','space','_index')
```

If `testuser` created indexes in the `writers` space, granting the following privileges is enough to alter indexes:

```
box.schema.user.grant('testuser','read','space','_space_sequence')
box.schema.user.grant('testuser','read,write','space','_index')
```

### CRUD operations

In this example, `testuser` gets privileges allowing them to *select data* from the „writers“ space:

```
box.schema.user.grant('testuser','read','space','writers')
```

In this example, `testuser` is allowed to read and modify data in the „books“ space:

```
box.schema.user.grant('testuser','read,write','space','books')
```

### Sequences

#### Creating and dropping sequences

In this example, `testuser` gets privileges to create *sequence* generators:

```
box.schema.user.grant('testuser','create','sequence')
box.schema.user.grant('testuser', 'read,write', 'space', '_sequence')
```

To let `testuser` drop a sequence, grant them the following privileges:

```
box.schema.user.grant('testuser','drop','sequence')
box.schema.user.grant('testuser','write','space','_sequence_data')
box.schema.user.grant('testuser','write','space','_sequence')
```

#### Using sequence functions

In this example, `testuser` is allowed to use the `id_seq:next()` function with a sequence named „id_seq“:

```
box.schema.user.grant('testuser','read,write','sequence','id_seq')
```

In the next example, `testuser` is allowed to use the `id_seq:set()` or `id_seq:reset()` functions with a sequence named „id_seq“:

```
box.schema.user.grant('testuser','write','sequence','id_seq')
```

### Functions

#### Creating and dropping functions

In this example, `testuser` gets privileges to create *functions*:

```
box.schema.user.grant('testuser','create','function')
box.schema.user.grant('testuser','read,write','space','_func')
```

To let `testuser` drop a function, grant them the following privileges:

```
box.schema.user.grant('testuser','drop','function')
box.schema.user.grant('testuser','write','space','_func')
```

#### Executing functions

To give the ability to execute a function named „sum“, grant the following privileges:

```
box.schema.user.grant('testuser','execute','function','sum')
```

#### Пользователи

In this example, `testuser` gets privileges to create other users:

```
box.schema.user.grant('testuser','create','user')
box.schema.user.grant('testuser', 'read,write', 'space', '_user')
box.schema.user.grant('testuser', 'write', 'space', '_priv')
```

#### Роли

To let `testuser` create new roles, grant the following privileges:

```
box.schema.user.grant('testuser','create','role')
box.schema.user.grant('testuser', 'read,write', 'space', '_user')
box.schema.user.grant('testuser', 'write', 'space', '_priv')
```

#### Executing code

To let `testuser` execute Lua code, grant the `execute` privilege to the `lua_eval` object:

```
box.schema.user.grant('testuser','execute','lua_eval')
```

Similarly, executing an arbitrary SQL expression requires the `execute` privilege to the `sql` object:

```
box.schema.user.grant('testuser','execute','sql')
```

#### Example

In the example below, the *created Lua function* is executed on behalf of its creator, even if called by another user.

First, the two spaces (`space1` and `space2`) are created, and a no-password user (`private_user`) is granted full access to them. Then `read_and_modify` is defined and `private_user` becomes this function's creator. Finally, another user (`public_user`) is granted access to execute Lua functions created by `private_user`.

```
box.schema.space.create('space1')
box.schema.space.create('space2')
box.space.space1:create_index('pk')
box.space.space2:create_index('pk')

box.schema.user.create('private_user')

box.schema.user.grant('private_user', 'read,write', 'space', 'space1')
box.schema.user.grant('private_user', 'read,write', 'space', 'space2')
box.schema.user.grant('private_user', 'create', 'universe')
box.schema.user.grant('private_user', 'read,write', 'space', '_func')

function read_and_modify(key)
  local space1 = box.space.space1
  local space2 = box.space.space2
  local fiber = require('fiber')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  local t = space1:get{key}
  if t ~= nil then
    space1:put{key, box.session.uid()}
    space2:put{key, fiber.time()}
  end
end

box.session.su('private_user')
box.schema.func.create('read_and_modify', {setuid= true})
box.session.su('admin')
box.schema.user.create('public_user', {password = 'secret'})
box.schema.user.grant('public_user', 'execute', 'function', 'read_and_modify')
```

Whenever `public_user` calls the function, it is executed on behalf of its creator, `private_user`.

### 6.6.5 All object types and permissions

#### Object types

| Object type | Description |
|---|---|
| `universe` | A database (*box.schema*) that contains database objects, including spaces, indexes, users, roles, sequences, and functions. Granting privileges to `universe` gives a user access to any object in the database. |
| `user` | A *user*. |
| `role` | A *role*. |
| `space` | A *space*. |
| `function` | A *function*. |
| `sequence` | A *sequence*. |
| `lua_eval` | Executing arbitrary Lua code. |
| `lua_call` | Calling any global user-defined Lua function. |
| `sql` | Executing an arbitrary SQL expression. |

### Permissions

| Permission | Object type | Granted to roles | Description |
|---|---|---|---|
| `read` | All | Yes | Allows reading data of the specified object. For example, this permission can be used to allow a user to select data from the specified space. |
| `write` | All | Yes | Allows updating data of the specified object. For example, this permission can be used to allow a user to modify data in the specified space. |
| `create` | All | Yes | Allows creating objects of the specified type. For example, this permission can be used to allow a user to create new spaces.<br>Note that this permission requires read and write access to certain system spaces. |
| `alter` | All | Yes | Allows altering objects of the specified type.<br>Note that this permission requires read and write access to certain system spaces. |
| `drop` | All | Yes | Allows dropping objects of the specified type.<br>Note that this permission requires read and write access to certain system spaces. |
| `execute` | `role`, `universe`, `function`, `lua_eval`, `lua_call`, `sql` | Yes | For `role`, allows using the specified role. For other object types, allows calling a function. |
| `session` | `universe` | No | Allows a user to connect to an instance over IPROTO. |
| `usage` | `universe` | No | Allows a user to use their privileges on database objects (for example, read, write, and alter spaces). |

### Object types and permissions

| Object type | Details |
|---|---|
| `universe` | • `read`: Allows reading any object types, including all spaces or sequence objects.<br>• `write`: Allows modifying any object types, including all spaces or sequence objects.<br>• `execute`: Allows execute functions, Lua code, or SQL expressions, including IPROTO calls.<br>• `session`: Allows a user to connect to an instance over IPROTO.<br>• `usage`: Allows a user to use their privileges on database objects (for example, read, write, and alter space).<br>• `create`: Allows creating users, roles, functions, spaces, and sequences. This permission requires read and write access to certain system spaces.<br>• `drop`: Allows creating users, roles, functions, spaces, and sequences. This permission requires read and write access to certain system spaces.<br>• `alter`: Allows altering user settings or space objects. |
| `user` | • `alter`: Allows modifying a user description, for example, change the password.<br>• `create`: Allows creating new users. This permission requires read and write access to the `_user` system space.<br>• `drop`: Allows dropping users. This permission requires read and write access to the `_user` system space. |
| `role` | • `execute`: Indicates that a role is assigned to the user or another role.<br>• `create`: Allows creating new roles. This permission requires read and write access to the `_user` system space.<br>• `drop`: Allows dropping roles. This permission requires read and write access to the `_user` system space. |
| `space` | • `read`: Allows selecting data from a space.<br>• `write`: Allows modifying data in a space.<br>• `create`: Allows creating new spaces. This permission requires read and write access to the `_space` system space.<br>• `drop`: Allows dropping spaces. This permission requires read and write access to the `_space` system space.<br>• `alter`: Allows modifying spaces. This permission requires read and write access to the `_space` system space.<br>If a space is created by a user, they can read and write it without granting explicit permission. |
| `function` | • `execute`: Allows calling a function.<br>• `create`: Allows creating a function. This permission requires read and write access to the `_func` system space.<br>If a function is created by a user, they can execute it without granting explicit permission.<br>• `drop`: Allows dropping a function. This permission requires read and write access to the `_func` system space. |
| `sequence` | • `read`: Allows using sequences in `space_obj:create_index()`.<br>• `write`: Allows all operations for a sequence object.<br>`seq_obj:drop()` requires a write permission to the `_priv` system space.<br>• `create`: Allows creating sequences. This permission requires read and write access to the `_sequence` system space.<br>If a sequence is created by a user, they can read/write it without explicit permission.<br>• `drop`: Allows dropping sequences. This permission requires read and write access to the `_sequence` system space.<br>• `alter`: Has no effect. `seq_obj:alter()` and other methods require the `write` |

## 6.7 Шардирование с vshard

Шардирование БД в Tarantool реализует модуль `vshard`. Вы можете обратиться к *руководству по быстрому запуску* этого модуля.

### 6.7.1 Установка

Модуль `vshard` не входит в основной дистрибутив Tarantool. Чтобы установить модуль, выполните команду:

```
$ tt rocks install vshard
```

**Примечание:** Для работы с модулем `vshard` необходимо установить: Tarantool версии 1.10.1 или выше, *пакет программ для разработки Tarantool*, `git`, `cmake` и `gcc`.

### 6.7.2 Настройка

Любой рабочий сегментированный кластер состоит из:

- one or more replica sets, each containing two or more *storage* instances,
- one or more *router* instances.

Количество хранилищ в наборе реплик определяет коэффициент избыточности данных. Рекомендуемое значение: 3 или более. Количество роутеров не ограничено, потому что у роутеров нет состояния. Рекомендуем увеличивать количество роутеров, если существующий экземпляр роутера ограничен возможностями процессора или ввода-вывода.

`vshard` поддерживает работу с несколькими роутерами в отдельном экземпляре Tarantool. Каждый роутер может подключиться к любому кластеру `vshard`. Несколько роутеров могут быть подключены к одному кластеру.

Поскольку приложения роутера (`router`) и хранилища (`storage`) выполняют совершенно разные наборы функций, их следует разворачивать на различных экземплярах Tarantool. Хотя технически возможно разместить приложение роутера на каждом узле типа хранилища, такой подход крайне не рекомендуется, и его следует избегать при развертывании в производственной среде.

Все хранилища можно развернуть, используя один набор файлов экземпляра (конфигурационных файлов).

Все роутеры также можно развернуть, используя один набор файлов экземпляра (конфигурационных файлов).

Топология всех узлов кластера должна быть одинаковой. Администратор должен убедиться, что конфигурации совпадают. Рекомендуем использовать инструмент управления конфигурациями, такой как Ansible или Puppet, во время развертывания кластера.

Шардинг не интегрирован ни в одну систему для централизованного управления конфигурациями. Предполагается, что само приложение отвечает за взаимодействие с такой системой и передачу параметров шардинга.

Пример настройки простого сегментированного кластера можно найти здесь.

### Вес реплики

Роутер отправляет все запросы чтения и записи только на мастер-экземпляр. Задав вес реплики, можно разрешить отправку запросов только на чтение не только на мастер-экземпляр, но и на доступную реплику, которая находится ближе всего к роутеру. Вес используется для определения расстояния между репликами в наборе реплик.

Например, вес можно использовать для определения физического расстояния между роутером и каждой репликой в наборе реплик. В таком случае запросы на чтение будут отправляться на ближайшую реплику (с наименьшим весом).

Кроме того, можно задать вес реплик, чтобы определить наиболее мощную реплику, которая может обрабатывать наибольшее количество запросов в секунду.

Основная идея состоит в том, чтобы указать зону для каждого `роутера` и каждой реплики, и таким образом составить матрицу относительных весов зоны. Этот подход позволяет устанавливать разный вес в разных зонах для одного набора реплик.

Чтобы задать вес, используйте атрибут zone (зона) для каждой реплики во время конфигурации:

```
local cfg = {
   sharding = {
      ['...uuid_набора_реплик...'] = {
         replicas = {
            ['...uuid_реплики...'] = {
                 ...,
                 zone = <число или строка>
            }
         }
      }
   }
}
```

Затем укажите относительный вес для каждой пары зон в параметре `weights` (вес) в `vshard.router.cfg`. Например:

```
weights = {
    [1] = {
        [2] = 1, -- Роутеры 1 зоны видят вес 2 зоны = 1.
        [3] = 2, -- Роутеры 1 зоны видят вес 3 зоны = 2 .
        [4] = 3, -- ...
    },
    [2] = {
        [1] = 10,
        [2] = 0,
        [3] = 10,
        [4] = 20,
    },
    [3] = {
        [1] = 100,
        [2] = 200, -- Роутеры 3 зоны видят вес 2 зоны = 200.
                   -- Обратите внимание, что этот вес не равен весу 2 зоны (= 2),
                   -- который видят роутеры 1 зоны (= 1).
        [4] = 1000,
    }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
}

local cfg = vshard.router.cfg({weights = weights, sharding = ...})
```

### Вес набора реплик

Вес набора реплик не равноценен весу реплики. Вес набора реплик определяет производительность набора реплик: чем больше вес, тем больше сегментов может хранить набор реплик. Общий размер всех сегментированных спейсов в наборе реплик также определяет его производительность.

Вес набора реплик можно рассматривать как относительный объем данных в наборе реплик. Например, если `replicaset_1 = 100`, и `replicaset_2 = 200`, второй набор реплик хранит в два раза больше сегментов, чем первый. По умолчанию веса всех наборов реплик равны.

Вес можно использовать, к примеру, чтобы хранить преобладающий объем данных в наборе реплик с большим объемом памяти.

### Процесс балансировки

Существует **эталонное число** сегментов в наборе реплик («эталонный» в данном случае значит идеальный). Если во всем наборе реплик это число остается неизменным, то сегменты распределяются равномерно.

Эталонное число рассчитывается автоматически с учетом количества сегментов в кластере и веса наборов реплик.

Балансировка начинается, когда **предел дисбаланса в наборе реплик** превышает предел дисбаланса, *указанный в конфигурации*.

Предел дисбаланса набора реплик рассчитывается следующим образом:

```
|эталонное_число_сегментов - текущее_число_сегментов| / эталонное_число_сегментов * 100
```

Например: Пользователь указал, что количество сегментов = 3000, а вес 3 наборов реплик составляет 1, 0,5 и 1,5. В результате получаем следующее эталонное число сегментов для наборов реплик: 1 набор реплик – 1000, 2 набор реплик – 500, 3 набор реплик – 1500.

Такой подход позволяет назначить нулевой вес для набора реплик, который запускает миграцию сегментов на оставшиеся узлы кластера. Это также позволяет добавить новый набор реплик с нулевой нагрузкой, который запускает миграцию сегментов из загруженных наборов реплик в набор реплик с нулевой нагрузкой.

---

**Примечание:** Новому набору реплик с нулевой нагрузкой следует присвоить вес, чтобы начать процесс балансировки.

---

При добавлении нового шарда конфигурацию можно обновить динамически:

1. Конфигурацию следует сначала обновить на всех роутерах, а затем на всех хранилищах.
2. Новый шард становится доступен для балансирования на уровне хранилища.
3. В результате балансировки происходит миграция сегментов на новый шард.
4. Если происходит запрос к перемещенному сегменту, `роутер` получает код ошибки с информацией о новом местонахождении сегмента.

В это время новый шард уже включен в пул соединений `роутера`, поэтому переадресация видима для приложения.

### Параллельная балансировка

Балансировщик в `vshard` первоначально был довольно прост: один процесс на одном узле, который рассчитывал *маршруты* отправки сегментов, сколько их отправлять и куда. Узлы применяли эти маршруты один за другим последовательно.

К сожалению, такая простая схема работала недостаточно быстро, особенно для Vinyl'а, где затраты ресурсов на чтение диска были сопоставимы с сетевыми затратами. На самом деле, механизм применения маршрутов в балансировщике Vinyl'а большую часть времени был в режиме ожидания.

Теперь каждый узел может параллельно посылать несколько сегментов по кругу в несколько пунктов назначения или всего в один.

Чтобы определять степень параллельности, используется новая опция *rebalancer_max_sending*. Задавать ее можно в конфигурации хранилища в корневой таблице:

```
cfg.rebalancer_max_sending = 5
vshard.storage.cfg(cfg, box.info.uuid)
```

Этот параметр не учитывается для роутеров.

---

**Примечание:** Задав `cfg.rebalancer_max_sending = N`, вы вряд ли получите N-кратное ускорение. На это влияют многие факторы: сеть, диск, количество других файберов в системе.

---

**Пример №1:**

> У вас уже есть 10 наборов реплик, добавили новый. Теперь все 10 наборов реплик будут пытаться отправить сегменты на новый.
>
> Предположим, каждый набор реплик может отправить до 5 сегментов одновременно. В этом случае будет довольно большая нагрузка на новый набор реплик: одновременная загрузка 50 сегментов. Если узлу нужно выполнить какую-то другую работу, возможно, такая большая нагрузка нежелательна. Кроме того, слишком большое количество параллельно загружаемых сегментов может привести к задержкам самого процесса балансировки.
>
> Чтобы исправить это, можно установить меньшее значение `rebalancer_max_sending` для старых наборов реплик или же уменьшить `rebalancer_max_receiving` для нового набора реплик. В последнем случае будет происходить управление загрузкой на старых узлах, и вы увидите это в логах.

Важно значение параметра `rebalancer_max_sending`, если у вас есть ограничение на максимальное количество сегментов, которые могут быть одновременно доступны только для чтения в кластере. Как упоминалось выше, во время отправки сегмент не принимает новые запросов на запись.

**Пример №2:**

> У вас есть 100 000 сегментов, и каждый сегмент хранит ~ 0,001% ваших данных. В кластере 10 наборов реплик. И нельзя позволить себе заблокировать для записи $>$ 0,1% данных. Таким образом, не следует устанавливать значение `rebalancer_max_sending` $>$ 10 на этих узлах. Тогда балансировщик не будет посылать более 100 сегментов одновременно по всему кластеру.

Если значение `max_sending` задано слишком высоко, а `max_receiving` слишком низко, то некоторые сегменты будут пытаться переместиться – и не смогут. При этом будут расходоваться сетевые ресурсы и время. Важно настроить эти параметры так, чтобы они не конфликтовали друг с другом.

### Блокировка набора реплик и закрепление корзины

Блокировка набора реплик делает набор реплик невидимым для балансировщика: заблокированный набор реплик не может ни принимать новые сегменты, ни мигрировать собственные сегменты.

В результате закрепления сегмента определенный сегмент блокируется для миграции: закрепленный сегмент остается в наборе реплик, в котором он закреплен, до отмены закрепления.

Закрепление всех сегментов в наборе реплик не означает блокирование набора реплик. Даже после закрепления всех сегментов незаблокированный набор реплик может принимать новые сегменты.

Блокировка набора реплик используется, к примеру, чтобы выделить для тестирования набор реплик из наборов реплик, используемых в производстве, или чтобы сохранить некоторые метаданные приложения, которые в течение некоторого времени не должны быть сегментированы. Закрепление сегмента используется в похожих случаях, но в меньшем масштабе.

Блокировка набора реплик и закрепление всех сегментов означает изоляцию целого набора реплик.

Заблокированные наборы реплик и закрепленные сегменты влияют на алгоритм балансировки, так как `балансировщик` должен игнорировать заблокированные наборы реплик и учитывать закрепленные сегменты при попытке достичь наилучшего возможного баланса.

Это нетривиальная задача, поскольку пользователь может закрепить слишком много сегментов в наборе реплик, так что становится невозможным достижение идеального баланса. Например, рассмотрим следующий кластер (предположим, что все веса наборов реплик равны 1).

Начальная конфигурация:

```
rs1: bucket_count = 150 -- число сегментов
rs2: bucket_count = 150, pinned_count = 120 -- число сегментов, число закрепленных␣
↪сегментов
```

Добавление нового набора реплик:

```
rs1: bucket_count = 150
rs2: bucket_count = 150, pinned_count = 120
rs3: bucket_count = 0
```

Идеальным балансом было бы `100 - 100 - 100`, чего невозможно достичь, поскольку набор реплик `rs2` содержит 120 закрепленных сегментов. The best possible balance here is the following:

```
rs1: bucket_count = 90
rs2: bucket_count = 120, pinned_count 120
rs3: bucket_count = 90
```

`Балансировщик` переместил максимально возможное количество сегментов из `rs2`, чтобы уменьшить дисбаланс. В то же время он учел одинаковый вес respected `rs1` и `rs3`.

Алгоритмы реализации блокировки и закрепления совершенно разные, хотя с точки зрения функций они похожи.

### Заблокированный набор реплик и балансировка

Заблокированные наборы реплик просто не участвуют в балансировке. Это означает, что даже если фактическое общее количество сегментов не равно эталонному числу, дисбаланс нельзя исправить из-за блокировки. Когда балансировщик обнаруживает, что один из наборов реплик заблокирован, он пересчитывает эталонное число сегментов неблокированных наборов реплик, как если бы заблокированный набор реплик и его сегменты вообще не существовали.

### Закрепленный набор реплик и балансировка

Балансировка наборов реплик с закрепленными сегментами требует более сложного алгоритма. Здесь `pinned_count[o]` – это число закрепленных сегментов, а `etalon_count` – это эталонное число сегментов для набора реплик:

1. Балансировщик рассчитывает эталонное число сегментов, как если бы все сегменты не были закреплены. Затем балансировщик проверяет каждый набор реплик и сопоставляет эталонное число сегментов с числом закрепленных сегментов в наборе реплик. Если `pinned_count < etalon_count`, незаблокированные наборы реплик (на данном этапе все заблокированные наборы реплик уже отфильтрованы) с закрепленными сегментами могут получать новые сегменты.
2. Если же `pinned_count > etalon_count`, дисбаланс исправить нельзя, так как балансировщик не может вывести закрепленные сегменты из этого набора реплик. В таком случае эталонное число обновляется как равное числу закрепленных сегментов. Наборы реплик с `pinned_count > etalon_count` не обрабатываются балансировщиком`, а число закрепленных сегментов вычитается из общего числа сегментов. Балансировщик пытается вывести как можно больше сегментов из таких наборов реплик.
3. Эта процедура перезапускается с шага 1 для наборов реплик с `pinned_count >= etalon_count` до тех пор, пока не будет выполнено условие `pinned_count <= etalon_count` для всех наборов реплик. Процедура также перезапускается при изменении общего числа сегментов.

Псевдокод для данного алгоритма будет следующим:

```
function cluster_calculate_perfect_balance(replicasets, bucket_count)
        -- балансировка сегментов с использованием веса рабочих наборов реплик --
end;

cluster = <all of the non-locked replica sets>;
bucket_count = <the total number of buckets in the cluster>;
can_reach_balance = false
while not can_reach_balance do
        can_reach_balance = true
        cluster_calculate_perfect_balance(cluster, bucket_count);
        foreach replicaset in cluster do
                if replicaset.perfect_bucket_count <
                   replicaset.pinned_bucket_count then
                        can_reach_balance = false
                        bucket_count -= replicaset.pinned_bucket_count;
                        replicaset.perfect_bucket_count =
                                replicaset.pinned_bucket_count;
                end;
        end;
end;
cluster_calculate_perfect_balance(cluster, bucket_count);
```

Сложность алгоритма составляет `O(N^2)`, где N – количество наборов реплик. На каждом шаге алгоритм либо завершает вычисление, либо игнорирует хотя бы один новый набор реплик, перегруженный закрепленными сегментами, и обновляет эталонное число сегментов в других наборах реплик.

### Ссылка в сегменте

Ссылка в сегменте – это счетчик в оперативной памяти, который похож на *закрепление сегмента* со следующими отличиями:

1. Ссылка в сегменте никогда не сохраняется. Ссылки предназначены для запрета передачи сегментов во время выполнения запроса, но при перезапуске все запросы отбрасываются.
2. Есть 2 типа ссылок в сегменте: только чтение (RO) и чтение-запись (RW).

   Если в сегменте есть ссылки типа RW, его нельзя перемещать. Однако, если балансировщику требуется отправка этого сегмента, он блокирует его для новых запросов на запись, ожидает завершения всех текущих запросов, а затем отправляет сегмент.

   Если в сегменте есть ссылки типа RO, его можно отправить, но нельзя удалить. Такой сегмент может даже перейти в статус мусора GARBAGE или отправки SENT, но его данные сохраняются до тех пор, пока не уйдет последний читатель.

   В одном сегменте могут быть ссылки как типа RO, так и типа RW.
3. Ссылки в сегменте исчисляются.

Методы *vshard.storage.bucket_ref/unref()* вызываются автоматически при использовании *vshard.router.call()* или *vshard.storage.call()*. При использовании API, например `r = vshard.router.route() r:callro/callrw`, следует дополнительно вызвать метод `bucket_ref()` в рамках функции. Кроме того, следует убедиться, что после `bucket_ref()` вызывается `bucket_unref()`, иначе сегмент нельзя перемещать из хранилища до перезапуска экземпляра.

Чтобы узнать количество ссылок в сегменте, используйте *vshard.storage.buckets_info([идентификатор_сегмента])* (параметр `идентификатор_сегмента` необязателен).

Пример:

```
vshard.storage.buckets_info(1)
---
- 1:
    status: active
    ref_rw: 1
    ref_ro: 1
    ro_lock: true
    rw_lock: true
    id: 1
```

### Определение спейса

Схема базы данных хранится на хранилищах, а роутеры ничего не знают о спейсах и кортежах.

В приложении хранилища следует определить спейсы с помощью `box.once()`. Например:

```
box.once("testapp:schema:1", function()
    local customer = box.schema.space.create('customer')
    customer:format({
        {'customer_id', 'unsigned'},
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        {'bucket_id', 'unsigned'},
        {'name', 'string'},
    })
    customer:create_index('customer_id', {parts = {'customer_id'}})
    customer:create_index('bucket_id', {parts = {'bucket_id'}, unique = false})

    local account = box.schema.space.create('account')
    account:format({
        {'account_id', 'unsigned'},
        {'customer_id', 'unsigned'},
        {'bucket_id', 'unsigned'},
        {'balance', 'unsigned'},
        {'name', 'string'},
    })
    account:create_index('account_id', {parts = {'account_id'}})
    account:create_index('customer_id', {parts = {'customer_id'}, unique = false})
    account:create_index('bucket_id', {parts = {'bucket_id'}, unique = false})
    box.snapshot()

    box.schema.func.create('customer_lookup')
    box.schema.role.grant('public', 'execute', 'function', 'customer_lookup')
    box.schema.func.create('customer_add')
end)
```

**Примечание:** В каждом спейсе, который вы планируете шардировать, должно быть поле с *идентификаторами сегментов*, проиндексированное с помощью *shard index*.

### Добавление данных

Все DML-операции с данными следует выполнять через роутер. Роутер поддерживает только вызов `CALL` через идентификатор сегмента `bucket_id`:

```
result = vshard.router.call(идентификатор_сегмента, режим, функция, аргументы)
```

`vshard.router.call()` направляет вызов `result = func(unpack(args))` на шард, который обслуживает идентификатор сегмента `bucket_id`.

Идентификатор сегмента `bucket_id` – это обычное число в диапазоне `1...`bucket_count<cfg_basic-bucket_count>». Этот номер можно произвольным образом назначить с помощью клиентского приложения. Сегментированный кластер Tarantool использует этот номер в качестве непрозрачного уникального идентификатора для распределения данных по множествам реплик. Мы гарантируем, что все записи с одним и тем же ``bucket_id` будут храниться в одном и том же наборе реплик.

### Настройка и перезапуск хранилища

В случае отказа мастера в наборе реплик рекомендуется:

1. Переключить одну из реплик в режим мастера, что позволит новому мастеру обрабатывать все входящие запросы.
2. Обновить конфигурацию всех членов кластера, в результате чего все запросы будут перенаправлены на новый мастер.

Мониторинг состояния мастера и переключение режимов экземпляров можно осуществлять с помощью внешней утилиты.

Для проведения запланированного остановки мастера в наборе реплик рекомендуется:

1. Обновить конфигурацию мастера и подождать синхронизации всех реплик, в результате чего все запросы будут перенаправлены на новый мастер.
2. Переключить другой экземпляр в режим мастера.
3. Обновить конфигурацию всех узлов.
4. Отключить старый мастер.

Для проведения запланированной остановки набора реплик рекомендуется:

1. Произвести миграцию всех сегментов в другие хранилища кластера.
2. Обновить конфигурацию всех узлов.
3. Отключить набор реплик.

В случае отказа всего набора реплик некоторая часть набора данных становится недоступной. Тем временем `роутер` пытается повторно подключиться к мастеру отказавшего набора реплик. Таким образом, после того, как набор реплик снова запущен, кластер автоматически восстанавливается.

### Файберы

Поиск сегментов, восстановление сегментов и балансировка сегментов выполняются автоматически и не требуют ручного вмешательства.

С технической точки зрения есть несколько файберов, которые отвечают за различные типы действий:

- файбер **обнаружения** на `роутере` выполняет поиск сегментов в фоновом режиме
- файбер **восстановления после отказа** на `роутере` поддерживает соединения с репликами
- файбер **сборки мусора** на каждом мастер-хранилище удаляет содержимое перемещенных сегментов
- файбер **восстановления сегмента** на каждом мастер-хранилище восстанавливает сегменты в статусах отправки SENDING и получения RECEIVING в случае перезагрузки
- **балансировщик** на отдельном мастер-хранилище среди множества наборов реплик выполняет процесс балансировки.

Для получения подробной информации см. разделы *Процесс балансировки* и *Миграция сегментов*.

#### Сборщик мусора

Файбер **сборщик мусора** работает в фоновом режиме на мастер-хранилищах в каждом наборе реплик. Он начинает удалять содержимое сегмента в состоянии мусора GARBAGE по частям. Когда сегмент пуст, запись о нем удаляется из системного спейса `_bucket`.

#### Восстановление сегмента

Файбер **восстановления сегмента** работает на мастер-хранилищах. Он помогает восстановить сегменты в статусах отправки SENDING и получения RECEIVING в случае перезагрузки.

Сегменты в статусе SENDING восстанавливаются следующим образом:

1. Сначала система ищет сегменты в статусе SENDING.
2. Если такой сегмент обнаружен, система отправляет запрос в целевой набор реплик.
3. Если сегмент в целевом наборе реплик находится в активном статусе ACTIVE, исходный сегмент удаляется из исходного узла.

Сегменты в статусе RECEIVING удаляются без дополнительных проверок.

#### Восстановление после отказа

Файбер **восстановления после отказа** работает на каждом роутере. Если мастер набора реплик становится недоступным, файбер перенаправляет запросы на чтение к репликам. Запросы на запись отклоняются с ошибкой до тех пор, пока мастер не будет доступен.

## 6.8 Replication administration

### 6.8.1 Мониторинг набора реплик

To learn what instances belong to the replica set and obtain statistics for all these instances, execute a *box.info.replication* request. The output below shows the replication status for a replica set containing one *master and two replicas*:

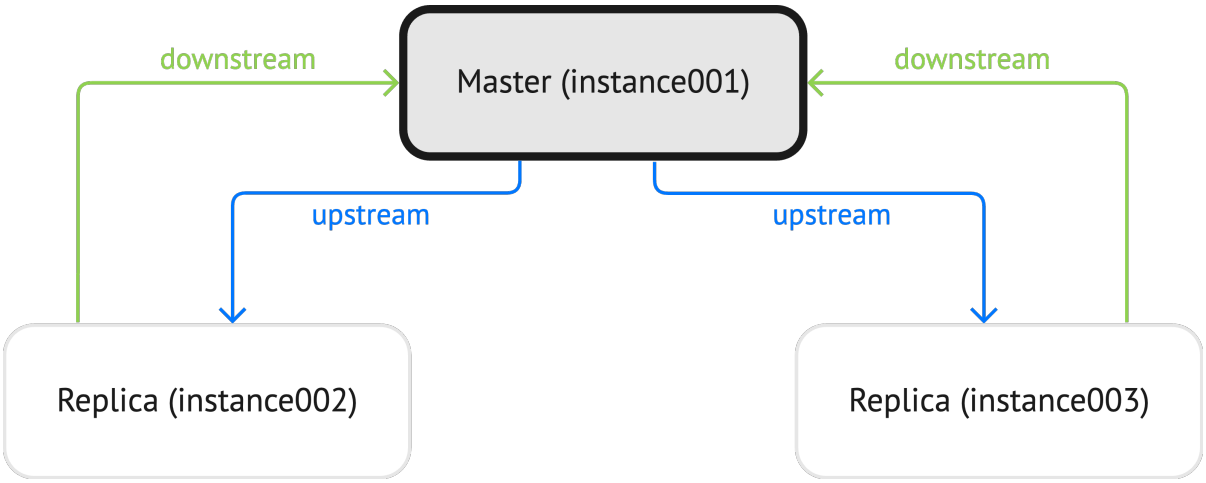
```
manual_leader:instance001> box.info.replication
---
- 1:
    id: 1
    uuid: 9bb111c2-3ff5-36a7-00f4-2b9a573ea660
    lsn: 21
    name: instance001
  2:
    id: 2
    uuid: 4cfa6e3c-625e-b027-00a7-29b2f2182f23
    lsn: 0
    upstream:
      status: follow
      idle: 0.052655000000414
      peer: replicator@127.0.0.1:3302
```

(continues on next page)

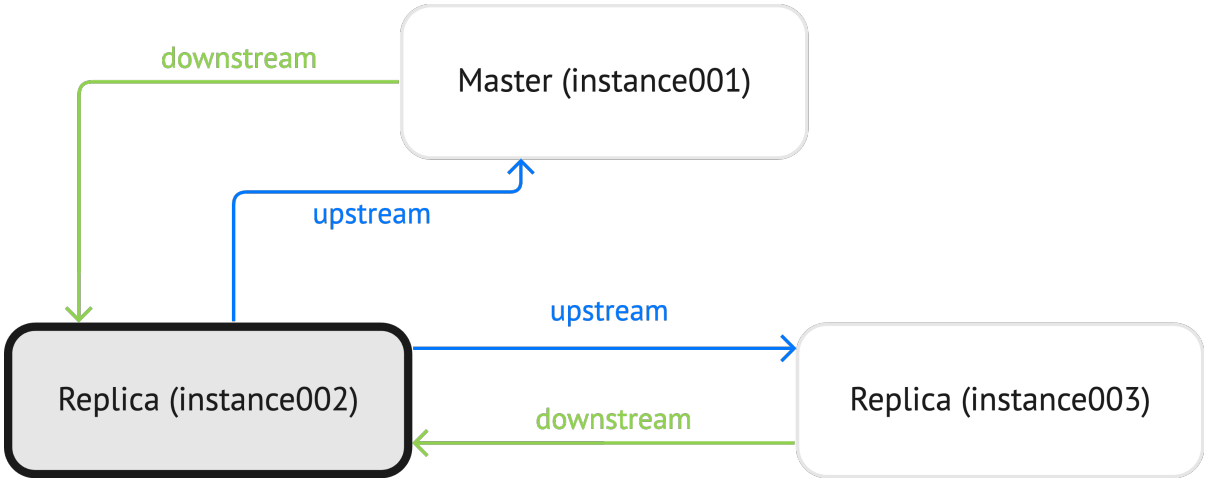
(продолжение с предыдущей страницы)

```
    lag: 0.00010204315185547
  name: instance002
  downstream:
    status: follow
    idle: 0.09503500000028
    vclock: {1: 21}
    lag: 0.00026917457580566
3:
  id: 3
  uuid: 9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6
  lsn: 0
  upstream:
    status: follow
    idle: 0.77522099999987
    peer: replicator@127.0.0.1:3303
    lag: 0.0001838207244873
  name: instance003
  downstream:
    status: follow
    idle: 0.33186100000012
    vclock: {1: 21}
    lag: 0
      ...
```

The following diagram illustrates the `upstream` and `downstream` connections if `box.info.replication` executed at the master instance (`instance001`):




If `box.info.replication` is executed on `instance002`, the `upstream` and `downstream` connections look as follows:




This means that statistics for replicas are given in regard to the instance on which `box.info.replication` is executed.

Основные индикаторы работоспособности репликации:

- *idle*: the time (in seconds) since the instance received the last event from a master.

  If the master has no updates to send to the replicas, it sends heartbeat messages every *replication_timeout* seconds. The master is programmed to disconnect if it does not see acknowledgments of the heartbeat messages within `replication_timeout` * 4 seconds.

  Таким образом, в работоспособном состоянии значение `idle` никогда не должно превышать значение `replication_timeout`: в противном случае, либо репликация сильно отстает, поскольку мастер опережает реплику, либо отсутствует сетевое подключение между экземплярами.

- *lag*: the time difference between the local time at the instance, recorded when the event was received, and the local time at another master recorded when the event was written to the *write-ahead log* on that master.

  Поскольку при расчете `отставания` используются часы операционной системы с двух разных машин, не удивляйтесь, получив отрицательное число: смещение во времени может привести к постоянному запаздыванию времени на удаленном мастере относительно часов на локальном экземпляре.

### 6.8.2 Восстановление после сбоя

«Сбой» – это ситуация, когда мастер становится недоступен вследствие проблем с оборудованием, сетевых неполадок или программной ошибки.

- In a master-replica set with *manual failover*, if a master disappears, error messages appear on the replicas stating that the connection is lost:

```
2023-12-04 13:19:04.724 [16755] main/110/applier/replicator@127.0.0.1:3301 I> can't␣
↪read row
2023-12-04 13:19:04.724 [16755] main/110/applier/replicator@127.0.0.1:3301 coio.
↪c:349 E> SocketError: unexpected EOF when reading from socket, called on fd 19,␣
↪aka 127.0.0.1:55932, peer of 127.0.0.1:3301: Broken pipe
2023-12-04 13:19:04.724 [16755] main/110/applier/replicator@127.0.0.1:3301 I> will␣
↪retry every 1.00 second
```

(continues on next page)

(продолжение с предыдущей страницы)

```
2023-12-04 13:19:04.724 [16755] relay/127.0.0.1:55940/101/main coio.c:349 E>␣
↪SocketError: unexpected EOF when reading from socket, called on fd 23, aka 127.0.
↪0.1:3302, peer of 127.0.0.1:55940: Broken pipe
2023-12-04 13:19:04.724 [16755] relay/127.0.0.1:55940/101/main I> exiting the relay␣
↪loop
```

- In a master-replica set with *automated failover*, a log also includes Raft messages showing the process of a new master's election:

```
2023-12-04 13:16:56.340 [16615] main/111/applier/replicator@127.0.0.1:3302 I> can't␣
↪read row
2023-12-04 13:16:56.340 [16615] main/111/applier/replicator@127.0.0.1:3302 coio.
↪c:349 E> SocketError: unexpected EOF when reading from socket, called on fd 24,␣
↪aka 127.0.0.1:55687, peer of 127.0.0.1:3302: Broken pipe
2023-12-04 13:16:56.340 [16615] main/111/applier/replicator@127.0.0.1:3302 I> will␣
↪retry every 1.00 second
2023-12-04 13:16:56.340 [16615] relay/127.0.0.1:55695/101/main coio.c:349 E>␣
↪SocketError: unexpected EOF when reading from socket, called on fd 25, aka 127.0.
↪0.1:3301, peer of 127.0.0.1:55695: Broken pipe
2023-12-04 13:16:56.340 [16615] relay/127.0.0.1:55695/101/main I> exiting the relay␣
↪loop
2023-12-04 13:16:59.690 [16615] main/112/applier/replicator@127.0.0.1:3303 I> RAFT:␣
↪message {term: 3, vote: 2, state: candidate, vclock: {1: 9}} from 2
2023-12-04 13:16:59.690 [16615] main/112/applier/replicator@127.0.0.1:3303 I> RAFT:␣
↪received a newer term from 2
2023-12-04 13:16:59.690 [16615] main/112/applier/replicator@127.0.0.1:3303 I> RAFT:␣
↪bump term to 3, follow
2023-12-04 13:16:59.690 [16615] main/112/applier/replicator@127.0.0.1:3303 I> RAFT:␣
↪vote for 2, follow
2023-12-04 13:16:59.691 [16615] main/119/raft_worker I> RAFT: persisted state
↪{term: 3}
2023-12-04 13:16:59.691 [16615] main/119/raft_worker I> RAFT: persisted state
↪{term: 3, vote: 2}
2023-12-04 13:16:59.691 [16615] main/112/applier/replicator@127.0.0.1:3303 I> RAFT:␣
↪message {term: 3, vote: 2, leader: 2, state: leader} from 2
2023-12-04 13:16:59.691 [16615] main/112/applier/replicator@127.0.0.1:3303 I> RAFT:␣
↪vote request is skipped - this is a notification about a vote for a third node,␣
↪not a request
2023-12-04 13:16:59.691 [16615] main/112/applier/replicator@127.0.0.1:3303 I> RAFT:␣
↪leader is 2, follow
```

The master's upstream status is reported as `disconnected` when executing *box.info.replication* on a replica:

```
auto_leader:instance001> box.info.replication
---
- 1:
    id: 1
    uuid: 4cfa6e3c-625e-b027-00a7-29b2f2182f23
    lsn: 32
    upstream:
      peer: replicator@127.0.0.1:3302
      lag: 0.00032305717468262
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      status: disconnected
      idle: 48.352504000002
      message: 'connect, called on fd 20, aka 127.0.0.1:62575: Connection␣
↪refused'
      system_message: Connection refused
    name: instance002
    downstream:
      status: stopped
      message: 'unexpected EOF when reading from socket, called on fd 32, aka␣
↪127.0.0.1:3301,
        peer of 127.0.0.1:62204: Broken pipe'
      system_message: Broken pipe
  2:
    id: 2
    uuid: 9bb111c2-3ff5-36a7-00f4-2b9a573ea660
    lsn: 1
    name: instance001
  3:
    id: 3
    uuid: 9a3a1b9b-8a18-baf6-00b3-a6e5e11fd8b6
    lsn: 0
    upstream:
      status: follow
      idle: 0.18620999999985
      peer: replicator@127.0.0.1:3303
      lag: 0.00012516975402832
    name: instance003
    downstream:
      status: follow
      idle: 0.19718099999955
      vclock: {2: 1, 1: 32}
      lag: 0.00051403045654297
...
```

To learn how to perform manual failover in a master-replica set, see the *Performing manual failover* section.

In a master-replica configuration with automated failover, a new master should be elected *automatically*.

### 6.8.3 Перезагрузка реплики

If any of a replica's write-ahead log or snapshot files are corrupted or deleted, you can reseed the replica. This procedure works only if the master's write-ahead logs are present.

1. Stop the replica using the *tt stop* command.
2. Delete write-ahead logs and snapshots stored in the `var/lib/<instance_name>` directory.

   **Примечание:** `var/lib` is the default directory used by tt to store write-ahead logs and snapshots. Learn more from *Configuration*.

3. Start the replica using the *tt start* command. The replica should catch up with the master by retrieving all the master's tuples.

4. (Optional) If you're reseeding a replica after a replication conflict, you also need to *restart replication*.

### 6.8.4 Решение конфликтов репликации

Tarantool guarantees that every update is applied only once on every replica. However, due to the asynchronous nature of replication, the order of updates is not guaranteed. This topic describes how to solve problems in *master-master* replication.

#### Замена по одному и тому же первичному ключу

**Case 1:** You have two instances of Tarantool. For example, you try to make a `replace` operation with the same primary key on both instances at the same time. This causes a conflict over which tuple to save and which one to discard.

*Триггер-функции* Тарантула могут помочь в реализации правил разрешения конфликтов при определенных условиях. Например, если у вас есть метка времени, то можно указать, что сохранять нужно кортеж с большей меткой.

First, you need a *before_replace()* trigger on the space which may have conflicts. In this trigger, you can compare the old and new replica records and choose which one to use (or skip the update entirely, or merge two records together).

Then you need to set the trigger at the right time before the space starts to receive any updates. The way you usually set the `before_replace` trigger is right when the space is created, so you need a trigger to set another trigger on the system space `_space`, to capture the moment when your space is created and set the trigger there. This can be an *on_replace()* trigger.

Разница между `before_replace` и `on_replace` заключается в том, что `on_replace` вызывается после вставки строки в спейс, а `before_replace` вызывается перед ней.

Устанавливать триггер `_space:on_replace()` также нужно в определенный момент. Лучшее время для его использования – это когда только что создан `_space`, что является триггером на *box.ctl.on_schema_init()*.

You also need to utilize `box.on_commit` to get access to the space being created. The resulting snippet would be the following:

```
local my_space_name = 'my_space'
local my_trigger = function(old, new) ... end -- ваша функция, устраняющая конфликт
box.ctl.on_schema_init(function()
    box.space._space:on_replace(function(old_space, new_space)
        if not old_space and new_space and new_space.name == my_space_name then
            box.on_commit(function()
                box.space[my_space_name]:before_replace(my_trigger)
            end
        end
    end)
end)
```

### Предотвращение дублирующей вставки

**Case 2:** In a replica set of two masters, both of them try to insert data by the same unique key:

```
tarantool> box.space.tester:insert{1, 'data'}
```

Это вызовет сообщение об ошибке дубликата ключа (`Duplicate key exists in unique index 'primary' in space 'tester'`), и репликация остановится. Такое поведение системы обеспечивается использованием рекомендуемого значения `false` (по умолчанию) для конфигурационного параметра *replication_skip_conflict*.

```
$ # сообщения об ошибках от мастера №1
2017-06-26 21:17:03.233 [30444] main/104/applier/rep_user@100.96.166.1 I> can't read row
2017-06-26 21:17:03.233 [30444] main/104/applier/rep_user@100.96.166.1 memtx_hash.cc:226␣
↪E> ER_TUPLE_FOUND:
Duplicate key exists in unique index 'primary' in space 'tester'
2017-06-26 21:17:03.233 [30444] relay/[::ffff:100.96.166.178]/101/main I> the replica␣
↪has closed its socket, exiting
2017-06-26 21:17:03.233 [30444] relay/[::ffff:100.96.166.178]/101/main C> exiting the␣
↪relay loop

$ # сообщения об ошибках от мастера №2
2017-06-26 21:17:03.233 [30445] main/104/applier/rep_user@100.96.166.1 I> can't read row
2017-06-26 21:17:03.233 [30445] main/104/applier/rep_user@100.96.166.1 memtx_hash.cc:226␣
↪E> ER_TUPLE_FOUND:
Duplicate key exists in unique index 'primary' in space 'tester'
2017-06-26 21:17:03.234 [30445] relay/[::ffff:100.96.166.178]/101/main I> the replica␣
↪has closed its socket, exiting
2017-06-26 21:17:03.234 [30445] relay/[::ffff:100.96.166.178]/101/main C> exiting the␣
↪relay loop
```

Если мы проверим статус репликации с помощью `box.info`, то увидим, что репликация на мастере №1 остановлена (`1.upstream.status = stopped`). Кроме того, данные с этого мастера не реплицируются (группа `1.downstream` отсутствует в отчете), поскольку встречается та же ошибка:

```
# статусы репликации (отчет от мастера №3)
tarantool> box.info
---
- version: 1.7.4-52-g980d30092
  id: 3
  ro: false
  vclock: {1: 9, 2: 1000000, 3: 3}
  uptime: 557
  lsn: 3
  vinyl: []
  cluster:
    uuid: 34d13b1a-f851-45bb-8f57-57489d3b3c8b
  pid: 30445
  status: running
  signature: 1000012
  replication:
    1:
      id: 1
      uuid: 7ab6dee7-dc0f-4477-af2b-0e63452573cf
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      lsn: 9
      upstream:
        peer: replicator@192.168.0.101:3301
        lag: 0.00050592422485352
        status: stopped
        idle: 445.8626639843
        message: Duplicate key exists in unique index 'primary' in space 'tester'
    2:
      id: 2
      uuid: 9afbe2d9-db84-4d05-9a7b-e0cbbf861e28
      lsn: 1000000
      upstream:
        status: follow
        idle: 201.99915885925
        peer: replicator@192.168.0.102:3301
        lag: 0.0015020370483398
      downstream:
        vclock: {1: 8, 2: 1000000, 3: 3}
    3:
      id: 3
      uuid: e826a667-eed7-48d5-a290-64299b159571
      lsn: 3
  uuid: e826a667-eed7-48d5-a290-64299b159571
...
```

To learn how to resolve a replication conflict by reseeding a replica, see *Resolving replication conflicts*.

### Replication runs out of sync

Предположим, что мы выполняем следующую операцию в кластере из двух экземпляров с конфигурацией мастер-мастер:

```
tarantool> box.space.tester:upsert({1}, {{'=', 2, box.info.uuid}})
```

Когда эта операция применяется на обоих экземплярах в наборе реплик:

```
# на мастере #1
tarantool> box.space.tester:upsert({1}, {{'=', 2, box.info.uuid}})
# на мастере #2
tarantool> box.space.tester:upsert({1}, {{'=', 2, box.info.uuid}})
```

... можно получить следующие результаты в зависимости от порядка выполнения:

- каждая строка мастера содержит UUID из мастера №1,
- каждая строка мастера содержит UUID из мастера №2,
- у мастера №1 UUID мастера №2, и наоборот.

### Commutative changes

The cases described in the previous paragraphs represent examples of **non-commutative** operations, that is operations whose result depends on the execution order. On the contrary, for **commutative operations**, the execution order does not matter.

Рассмотрим, например, следующую команду:

```
tarantool> box.space.tester:upsert{{1, 0}, {{'+', 2, 1)}
```

Эта операция коммутативна: получаем одинаковый результат, независимо от порядка, в котором обновление применяется на других мастерах.

### Trigger usage

The logic and the snippet setting a trigger will be the same here as in *case 1*. But the trigger function will differ. Note that the trigger below assumes that tuple has a timestamp in the second field.

```
local my_space_name = 'test'
local my_trigger = function(old, new, sp, op)
    -- op:  'INSERT', 'DELETE', 'UPDATE', or 'REPLACE'
    if new == nil then
        print("No new during "..op, old)
        return -- удаление допустимо
    end
    if old == nil then
        print("Insert new, no old", new)
        return new  -- вставка без старого значения допустима
    end
    print(op.." duplicate", old, new)
    if op == 'INSERT' then
        if new[2] > old[2] then
            -- Создание нового кортежа сменит оператор на REPLACE
            return box.tuple.new(new)
        end
        return old
    end
    if new[2] > old[2] then
        return new
    else
        return old
    end
    return
end

box.ctl.on_schema_init(function()
    box.space._space:on_replace(function(old_space, new_space)
        if not old_space and new_space and new_space.name == my_space_name then
            box.on_commit(function()
                box.space[my_space_name]:before_replace(my_trigger)
            end)
        end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    end)
end)
```

## 6.9 Просмотр состояния сервера

### 6.9.1 Использование Tarantool в качестве клиента

Tarantool входит в интерактивный режим, если:

- вы запускаете его без *файла экземпляра*, либо
- в файле экземпляра содержится команда *console.start()*.

Tarantool выводит приглашение командной строки (например, «tarantool>») – и вы можете посылать запросы. Если использовать Tarantool таким образом, он может выступать клиентом для удаленного сервера, см. простые примеры в *Руководстве для начинающих*.

The interactive mode is used in the `tt` utility's *connect* command.

### 6.9.2 Выполнение кода на экземпляре Tarantool

You can attach to an instance's *admin console* and execute some Lua code using `tt`:

```
$ # for local instances:
$ tt connect my_app
   • Connecting to the instance...
   • Connected to /var/run/tarantool/example.control

/var/run/tarantool/my_app.control> 1 + 1
---
- 2
...
/var/run/tarantool/my_app.control>

$ # for local and remote instances:
$ tt connect username:password@127.0.0.1:3306
```

You can also use `tt` to execute Lua code on an instance without attaching to its admin console. For example:

```
$ # executing commands directly from the command line
$ <command> | tt connect my_app -f -
<...>

$ # - OR -

$ # executing commands from a script file
$ tt connect my_app -f script.lua
<...>
```

**Примечание:** Alternatively, you can use the *console* module or the *net.box* module from a Tarantool server. Also, you can write your client programs with any of the *connectors*. However, most of the examples

in this manual illustrate usage with either `tt connect` or *using the Tarantool server as a client*.

### 6.9.3 Проверка состояния экземпляра

To check the instance status, run:

```
$ tt status my_app
INSTANCE     STATUS      PID
my_app       RUNNING     67172

$ # - OR -

$ systemctl status tarantool@my_app
```

To check the boot log, on systems with `systemd`, run:

```
$ journalctl -u tarantool@my_app -n 5
```

For more specific checks, use the reports provided by functions in the following submodules:

- *Вложенный модуль box.cfg* – проверка и указание всех конфигурационных параметров Tarantool-сервера,
- *Вложенный модуль box.slab* – мониторинг использования и фрагментированности памяти, выделенной для хранения данных в Tarantool,
- *Вложенный модуль box.info* – просмотр переменных Tarantool-сервера – в первую очередь тех, что относятся к репликации,
- *Вложенный модуль box.stat* – просмотр статистики Tarantool по запросам и использованию сети,

Finally, there is the metrics library, which enables collecting metrics (such as memory usage or number of requests) from Tarantool applications and expose them via various protocols, including Prometheus. Check *Monitoring* for more details.

**Пример**

Очень часто администраторам приходится вызывать функцию *box.slab.info()*, которая показывает подробную статистику по использованию памяти для конкретного экземпляра Tarantool.

```
tarantool> box.slab.info()
---
- items_size: 228128
  items_used_ratio: 1.8%
  quota_size: 1073741824
  quota_used_ratio: 0.8%
  arena_used_ratio: 43.2%
  items_used: 4208
  quota_used: 8388608
  arena_size: 2325176
  arena_used: 1003632
...
```

Tarantool занимает память операционной системы, например, когда пользователь вставляет много данных. Можно проверить, сколько памяти занято, выполнив команду (в Linux):

```
ps -eo args,%mem | grep "tarantool"
```

Tarantool почти никогда не освобождает эту память, даже если пользователь удалит все, что было вставлено, или уменьшит фрагментацию, вызвав сборщик мусора в Lua с помощью функции collectgarbage.

Как правило, это не влияет на производительность. Однако, чтобы заставить Tarantool высвободить память, можно вызвать :*box.snapshot()*, остановить экземпляр и перезапустить его.

### 6.9.4 Inspect traffic

Inspecting binary traffic is a boring task. We offer a Wireshark plugin to simplify the analysis of Tarantool's traffic.

To enable the plugin, follow the steps below.

Clone the tarantool-dissector repository:

```
git clone https://github.com/tarantool/tarantool-dissector.git
```

Copy or symlink the plugin files into the Wireshark plugin directory:

```
mkdir -p ~/.local/lib/wireshark/plugins
cd ~/.local/lib/wireshark/plugins
ln -s /path/to/tarantool-dissector/MessagePack.lua ./
ln -s /path/to/tarantool-dissector/tarantool.dissector.lua ./
```

(For the location of the plugin directory on macOS and Windows, please refer to the Plugin folders chapter in the Wireshark documentation.)

Run the Wireshark GUI and ensure that the plugins are loaded:

- Open *Help* > *About Wireshark* > *Plugins*.
- Find `MessagePack.lua` and `tarantool.dissector.lua` in the list.

Now you can inspect incoming and outgoing Tarantool packets with user-friendly annotations.

Visit the project page for details: https://github.com/tarantool/tarantool-dissector.

### 6.9.5 Профилирование производительности

Иногда Tarantool может работать медленнее, чем обычно. Причин такого поведения может быть несколько: проблемы с диском, Lua-скрипты, активно использующие процессор, или неправильная настройка. В таких случаях в журнале Tarantool'а могут отсутствовать необходимые подробности, поэтому единственным признаком неправильного поведения является наличие в журнале записей вида `W> too long DELETE: 8.546 sec`. Ниже приведены инструменты и приемы, которые облегчают снятие профиля производительности Tarantool'а. Эта процедура может помочь при решении проблем с замедлением.

---

**Примечание:** Большинство инструментов, за исключением `fiber.info()`, предназначено для дистрибутивов GNU/Linux, но не для FreeBSD или Mac OS.

---

### fiber.info()

Самый простой способ профилирования – это использование встроенных функций Tarantool'а. *fiber.info()* возвращает информацию обо всех работающих файберах с соответствующей трассировкой стека для языка C. Эти данные показывают, сколько файберов запущенно на данный момент и какие функции, написанные на C, вызываются чаще остальных.

Сначала войдите в интерактивную административную консоль вашего экземпляра Tarantool'а:

```
$ tt connect NAME|URI
```

После этого загрузите модуль `fiber`:

```
tarantool> fiber = require('fiber')
```

Теперь можно получить необходимую информацию с помощью `fiber.info()`.

На этом шаге в вашей консоли должно выводиться следующее:

```
tarantool> fiber = require('fiber')
---
...
tarantool> fiber.info()
---
- 360:
    csw: 2098165
    backtrace:
    - '#0 0x4d1b77 in wal_write(journal*, journal_entry*)+487'
    - '#1 0x4bbf68 in txn_commit(txn*)+152'
    - '#2 0x4bd5d8 in process_rw(request*, space*, tuple**)+136'
    - '#3 0x4bed48 in box_process1+104'
    - '#4 0x4d72f8 in lbox_replace+120'
    - '#5 0x50f317 in lj_BC_FUNCC+52'
    fid: 360
    memory:
      total: 61744
      used: 480
    name: main
  129:
    csw: 113
    backtrace: []
    fid: 129
    memory:
      total: 57648
      used: 0
    name: 'console/unix/:'
...
```

Мы рекомендуем присваивать создаваемым файберам понятные имена, чтобы их можно было легко найти в списке, выводимом `fiber.info()`. В примере ниже создается файбер с именем `myworker`:

```
tarantool> fiber = require('fiber')
---
...
tarantool> f = fiber.create(function() while true do fiber.sleep(0.5) end end)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...
tarantool> f:name('myworker') <!-- присваивание имени файберу
---
...
tarantool> fiber.info()
---
- 102:
    csw: 14
    backtrace:
    - '#0 0x501a1a in fiber_yield_timeout+90'
    - '#1 0x4f2008 in lbox_fiber_sleep+72'
    - '#2 0x5112a7 in lj_BC_FUNCC+52'
    fid: 102
    memory:
      total: 57656
      used: 0
    name: myworker <!-- новый созданный фоновый файбер
  101:
    csw: 284
    backtrace: []
    fid: 101
    memory:
      total: 57656
      used: 0
    name: interactive
...
```

Для принудительного завершения файбера используется команда *fiber.kill(fid)*:

```
tarantool> fiber.kill(102)
---
...
tarantool> fiber.info()
---
- 101:
    csw: 324
    backtrace: []
    fid: 101
    memory:
      total: 57656
      used: 0
    name: interactive
...
```

Чтобы получить таблицу всех рабочих файберов, можно использовать *fiber.top()*.

Если вам необходимо динамически получать информацию с помощью `fiber.info()`, вам может пригодиться приведенный ниже скрипт. Он каждые полсекунды подключается к экземпляру Tarantool'а, указанному в переменной `NAME`, выполняет команду `fiber.info()` и записывает ее выход в файл `fiber-info.txt`:

```
$ rm -f fiber.info.txt
$ watch -n 0.5 "echo 'require(\"fiber\").info()' | tt connect NAME -f - | tee -a fiber-
↪info.txt"
```

Если вы не можете самостоятельно разобраться, какой именно файбер вызывает проблемы с производительностью, запустите данный скрипт на 10-15 секунд и пришлите получившийся файл команде Tarantool'а на адрес support@tarantool.org.

### Простейшие профилировщики

**pstack <pid>**

Чтобы использовать этот инструмент, его необходимо установить с помощью пакетного менеджера, поставляемого с вашим дистрибутивом Linux. Данная команда выводит трассировку стека выполнения для работающего процесса с соответствующим PID. При необходимости команду можно запустить несколько раз, чтобы выявить узкое место, которое вызывает падение производительности.

После установки воспользуйтесь следующей командой:

```
$ pstack $(pidof tarantool INSTANCENAME.lua)
```

Затем выполните:

```
$ echo $(pidof tarantool INSTANCENAME.lua)
```

чтобы вывести на экран PID экземпляра Tarantool'а, использующего файл `INSTANCENAME.lua`.

В вашей консоли должно отображаться приблизительно следующее:

```
Thread 19 (Thread 0x7f09d1bff700 (LWP 24173)):
#0 0x00007f0a1a5423f2 in ?? () from /lib64/libgomp.so.1
#1 0x00007f0a1a53fdc0 in ?? () from /lib64/libgomp.so.1
#2 0x00007f0a1ad5adc5 in start_thread () from /lib64/libpthread.so.0
#3 0x00007f0a1a050ced in clone () from /lib64/libc.so.6
Thread 18 (Thread 0x7f09d13fe700 (LWP 24174)):
#0 0x00007f0a1a5423f2 in ?? () from /lib64/libgomp.so.1
#1 0x00007f0a1a53fdc0 in ?? () from /lib64/libgomp.so.1
#2 0x00007f0a1ad5adc5 in start_thread () from /lib64/libpthread.so.0
#3 0x00007f0a1a050ced in clone () from /lib64/libc.so.6
<...>
Thread 2 (Thread 0x7f09c8bfe700 (LWP 24191)):
#0 0x00007f0a1ad5e6d5 in pthread_cond_wait@@GLIBC_2.3.2 () from /lib64/libpthread.so.0
#1 0x0000000000045d901 in wal_writer_pop(wal_writer*) ()
#2 0x0000000000045db01 in wal_writer_f(__va_list_tag*) ()
#3 0x0000000000429abc in fiber_cxx_invoke(int (*)(__va_list_tag*), __va_list_tag*) ()
#4 0x00000000004b52a0 in fiber_loop ()
#5 0x00000000006099cf in coro_init ()
Thread 1 (Thread 0x7f0a1c47fd80 (LWP 24172)):
#0 0x00007f0a1a0512c3 in epoll_wait () from /lib64/libc.so.6
#1 0x00000000006051c8 in epoll_poll ()
#2 0x0000000000607533 in ev_run ()
#3 0x0000000000428e13 in main ()
```

**gdb -ex «bt» -p <pid>**

Как и в случае с `pstack`, перед использованием GNU-отладчик (также известный как `gdb`) необходимо сначала установить через пакетный менеджер, встроенный в ваш дистрибутив Linux.

После установки воспользуйтесь следующей командой:

```
$ gdb -ex "set pagination 0" -ex "thread apply all bt" --batch -p $(pidof tarantool␣
↪INSTANCENAME.lua)
```

Затем выполните:

```
$ echo $(pidof tarantool INSTANCENAME.lua)
```

чтобы вывести на экран PID экземпляра Tarantool'а, использующего файл `INSTANCENAME.lua`.

После использования отладчика в консоль должна выводиться следующая информация:

```
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

[CUT]

Thread 1 (Thread 0x7f72289ba940 (LWP 20535)):
#0 _int_malloc (av=av@entry=0x7f7226e0eb20 <main_arena>, bytes=bytes@entry=504) at␣
↪malloc.c:3697
#1 0x00007f7226acf21a in __libc_calloc (n=<optimized out>, elem_size=<optimized out>) at␣
↪malloc.c:3234
#2 0x00000000004631f8 in vy_merge_iterator_reserve (capacity=3, itr=0x7f72264af9e0) at /
↪usr/src/tarantool/src/box/vinyl.c:7629
#3 vy_merge_iterator_add (itr=itr@entry=0x7f72264af9e0, is_mutable=is_mutable@entry=true,
↪ belong_range=belong_range@entry=false) at /usr/src/tarantool/src/box/vinyl.c:7660
#4 0x000000000047703df in vy_read_iterator_add_mem (itr=0x7f72264af990) at /usr/src/
↪tarantool/src/box/vinyl.c:8387
#5 vy_read_iterator_use_range (itr=0x7f72264af990) at /usr/src/tarantool/src/box/vinyl.
↪c:8453
#6 0x000000000047657d in vy_read_iterator_start (itr=<optimized out>) at /usr/src/
↪tarantool/src/box/vinyl.c:8501
#7 0x000000000047766b5 in vy_read_iterator_next (itr=itr@entry=0x7f72264af990,␣
↪result=result@entry=0x7f72264afad8) at /usr/src/tarantool/src/box/vinyl.c:8592
#8 0x000000000047689d in vy_index_get (tx=tx@entry=0x7f7226468158,␣
↪index=index@entry=0x2563860, key=<optimized out>, part_count=<optimized out>,␣
↪result=result@entry=0x7f72264afad8) at /usr/src/tarantool/src/box/vinyl.c:5705
#9 0x0000000000477601 in vy_replace_impl (request=<optimized out>, request=<optimized␣
↪out>, stmt=0x7f72265a7150, space=0x2567ea0, tx=0x7f7226468158) at /usr/src/tarantool/
↪src/box/vinyl.c:5920
#10 vy_replace (tx=0x7f7226468158, stmt=stmt@entry=0x7f72265a7150, space=0x2567ea0,␣
↪request=<optimized out>) at /usr/src/tarantool/src/box/vinyl.c:6608
#11 0x00000000004615a9 in VinylSpace::executeReplace (this=<optimized out>, txn=
↪<optimized out>, space=<optimized out>, request=<optimized out>) at /usr/src/tarantool/
↪src/box/vinyl_space.cc:108
#12 0x00000000004bd723 in process_rw (request=request@entry=0x7f72265a70f8,␣
↪space=space@entry=0x2567ea0, result=result@entry=0x7f72264afbc8) at /usr/src/tarantool/
↪src/box/box.cc:182
#13 0x00000000004bed48 in box_process1 (request=0x7f72265a70f8,␣
↪result=result@entry=0x7f72264afbc8) at /usr/src/tarantool/src/box/box.cc:700
#14 0x00000000004bf389 in box_replace (space_id=space_id@entry=513, tuple=<optimized out>
↪, tuple_end=<optimized out>, result=result@entry=0x7f72264afbc8) at /usr/src/tarantool/
↪src/box/box.cc:754
```

(continues on next page)

(продолжение с предыдущей страницы)

```
#15 0x00000000004d72f8 in lbox_replace (L=0x413c5780) at /usr/src/tarantool/src/box/lua/
↪index.c:72
#16 0x000000000050f317 in lj_BC_FUNCC ()
#17 0x00000000004d37c7 in execute_lua_call (L=0x413c5780) at /usr/src/tarantool/src/box/
↪lua/call.c:282
#18 0x000000000050f317 in lj_BC_FUNCC ()
#19 0x0000000000529c7b in lua_cpcall ()
#20 0x00000000004f6aa3 in luaT_cpcall (L=L@entry=0x413c5780, func=func@entry=0x4d36d0
↪<execute_lua_call>, ud=ud@entry=0x7f72264afde0) at /usr/src/tarantool/src/lua/utils.
↪c:962
#21 0x00000000004d3fe7 in box_process_lua (handler=0x4d36d0 <execute_lua_call>,␣
↪out=out@entry=0x7f7213020600, request=request@entry=0x413c5780) at /usr/src/tarantool/
↪src/box/lua/call.c:382
#22 box_lua_call (request=request@entry=0x7f72130401d8, out=out@entry=0x7f7213020600) at␣
↪/usr/src/tarantool/src/box/lua/call.c:405
#23 0x00000000004c0f27 in box_process_call (request=request@entry=0x7f72130401d8,␣
↪out=out@entry=0x7f7213020600) at /usr/src/tarantool/src/box/box.cc:1074
#24 0x000000000041326c in tx_process_misc (m=0x7f7213040170) at /usr/src/tarantool/src/
↪box/iproto.cc:942
#25 0x0000000000504554 in cmsg_deliver (msg=0x7f7213040170) at /usr/src/tarantool/src/
↪cbus.c:302
#26 0x0000000000504c2e in fiber_pool_f (ap=<error reading variable: value has been␣
↪optimized out>) at /usr/src/tarantool/src/fiber_pool.c:64
#27 0x000000000041122c in fiber_cxx_invoke(fiber_func, typedef __va_list_tag __va_list_
↪tag *) (f=<optimized out>, ap=<optimized out>) at /usr/src/tarantool/src/fiber.h:645
#28 0x00000000005011a0 in fiber_loop (data=<optimized out>) at /usr/src/tarantool/src/
↪fiber.c:641
#29 0x0000000000688fbf in coro_init () at /usr/src/tarantool/third_party/coro/coro.c:110
```

Запустите отладчик в цикле, чтобы собрать достаточно информации, которая поможет установить причину спада производительности Tarantool'а. Можно воспользоваться следующим скриптом:

```
$ rm -f stack-trace.txt
$ watch -n 0.5 "gdb -ex 'set pagination 0' -ex 'thread apply all bt' --batch -p $(pidof␣
↪tarantool INSTANCENAME.lua) | tee -a stack-trace.txt"
```

С точки зрения структуры и функциональности, этот скрипт идентичен тому, что используется выше с `fiber.info()`.

Если вам не удается отыскать причину пониженной производительности, запустите данный скрипт на 10-15 секунд и пришлите получившийся файл `stack-trace.txt` команде Tarantool'а на адрес support@tarantool.org.

**Предупреждение:** Следует использовать `pstack` и `gdb` с осторожностью: каждый раз, подключаясь с работающему процессу, они приостанавливают выполнение этого процесса приблизительно на одну секунду, что может иметь серьезные последствия для высоконагруженных сервисов.

### gperftools

Чтобы использовать профилировщик процессора из набора Google Performance Tools с Tarantool'ом, необходимо сначала установить зависимости:

- Если вы используете Debian/Ubuntu, запустите эту команду:

```
$ apt-get install libgoogle-perftools4
```

- Если вы используете RHEL/CentOS/Fedora, запустите эту команду:

```
$ yum install gperftools-libs
```

После этого установите привязки для Lua:

```
$ tt rocks install gperftools
```

После окончания установки войдите в интерактивную административную консоль вашего экземпляра Tarantool'а:

```
$ tt connect NAME|URI
```

Для запуска профилировщика выполните следующий код:

```
tarantool> cpuprof = require('gperftools.cpu')
tarantool> cpuprof.start('/home/<имя_пользователя>/tarantool-on-production.prof')
```

На сбор метрик производительности у профилировщика уходит по крайней мере пара минут. По истечении этого времени можно сохранять информацию на диск (неограниченное количество раз):

```
tarantool> cpuprof.flush()
```

Для остановки профилировщика выполните следующую команду:

```
tarantool> cpuprof.stop()
```

Теперь можно проанализировать собранные данные с помощью утилиты `pprof`, которая входит в пакет `gperftools`:

```
$ pprof --text /usr/bin/tarantool /home/<имя_пользователя>/tarantool-on-production.prof
```

---

**Примечание:** В дистрибутивах Debian/Ubuntu утилита `pprof` называется `google-pprof`.

---

В консоль должно выводиться приблизительно следующее:

```
Total: 598 samples
      83 13.9% 13.9% 83 13.9% epoll_wait
      54 9.0% 22.9% 102 17.1%
vy_mem_tree_insert.constprop.35
      32 5.4% 28.3% 34 5.7% __write_nocancel
      28 4.7% 32.9% 42 7.0% vy_mem_iterator_start_from
      26 4.3% 37.3% 26 4.3% _IO_str_seekoff
      21 3.5% 40.8% 21 3.5% tuple_compare_field
      19 3.2% 44.0% 19 3.2%
```

(continues on next page)

(продолжение с предыдущей страницы)

```
::TupleCompareWithKey::compare
      19 3.2% 47.2% 38 6.4% tuple_compare_slowpath
      12 2.0% 49.2% 23 3.8% __libc_calloc
       9 1.5% 50.7% 9 1.5%
::TupleCompare::compare@42efc0
       9 1.5% 52.2% 9 1.5% vy_cache_on_write
       9 1.5% 53.7% 57 9.5% vy_merge_iterator_next_key
       8 1.3% 55.0% 8 1.3% __nss_passwd_lookup
       6 1.0% 56.0% 25 4.2% gc_onestep
       6 1.0% 57.0% 6 1.0% lj_tab_next
       5 0.8% 57.9% 5 0.8% lj_alloc_malloc
       5 0.8% 58.7% 131 21.9% vy_prepare
```

### perf

Этот инструмент для мониторинга и анализа производительности устанавливается отдельно с помощью пакетного менеджера. Попробуйте ввести в окне консоли команду `perf` и следуйте подсказкам, чтобы установить необходимые пакеты.

---

**Примечание:** По умолчанию некоторые команды из пакета `perf` можно выполнять только с **root**-правами, поэтому необходимо либо зайти в систему из-под пользователя **root**, либо добавлять перед каждой командой `sudo`.

---

Чтобы начать сбор показателей производительности, выполните следующую команду:

```
$ perf record -g -p $(pidof tarantool INSTANCENAME.lua)
```

Эта команда сохраняет собранные данные в файл `perf.data`, который находится в текущей рабочей папке. Для остановки процесса (обычно через 10-15 секунд) нажмите **ctrl**+**C**. В консоли должно появиться следующее:

```
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.225 MB perf.data (1573 samples) ]
```

Затем выполните эту команду:

```
$ perf report -n -g --stdio | tee perf-report.txt
```

Она превращает содержащиеся в `perf.data` статистические данные в отчет о производительности, который сохраняется в файл `perf-report.txt`.

Получившийся отчет выглядит следующим образом:

```
# Samples: 14K of event 'cycles'
# Event count (approx.): 9927346847
#
# Children Self Samples Command Shared Object Symbol
# ........ ........ ............ ......... ................... ........................
↪.............
#
    35.50% 0.55% 79 tarantool tarantool [.] lj_gc_step
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            |
             --34.95%--lj_gc_step
                       |
                       |--29.26%--gc_onestep
                       | |
                       | |--13.85%--gc_sweep
                       | | |
                       | | |--5.59%--lj_alloc_free
                       | | |
                       | | |--1.33%--lj_tab_free
                       | | | |
                       | | | --1.01%--lj_alloc_free
                       | | |
                       | | --1.17%--lj_cdata_free
                       | |
                       | |--5.41%--gc_finalize
                       | | |
                       | | |--1.06%--lj_obj_equal
                       | | |
                       | | --0.95%--lj_tab_set
                       | |
                       | |--4.97%--rehashtab
                       | | |
                       | | --3.65%--lj_tab_resize
                       | | |
                       | | |--0.74%--lj_tab_set
                       | | |
                       | | --0.72%--lj_tab_newkey
                       | |
                       | |--0.91%--propagatemark
                       | |
                       | --0.67%--lj_cdata_free
                       |
                        --5.43%--propagatemark
                                  |
                                   --0.73%--gc_mark
```

Инструменты `gperftools` и `perf` отличаются от `pstack` и `gdb` низкой затратой ресурсов (пренебрежимо малой по сравнению с `pstack` и `gdb`): они подключаются к работающим процессам без больших задержек, а потому могут использоваться без серьезных последствий.

### jit.p

Профилировщик «jit.p» входит в комплект сервера приложений Tarantool. Чтобы загрузить его, выполните команду `require('jit.p')` или `require('jit.profile')`. Есть много параметров для настройки выборки и вывода, они описаны в документации по профилировщику LuaJIT, которая доступна в репозитории LuaJIT на GitHub в ветке 2.1 в файле: `doc/ext_profiler.html`.

#### Пример

Создайте функцию для вызова функции под названием f1, которая осуществляет 500 000 вставок и удалений в спейсе Tarantool. Запустите профилировщик, выполните функцию, завершите работу профилировщика. Получите результат выборки профилировщика.

```
box.space.t:drop()
box.schema.space.create('t')
box.space.t:create_index('i')
function f1() for i = 1,500000 do
  box.space.t:insert{i}
  box.space.t:delete{i}
  end
return 1
end
function f3() f1() end
jit_p = require("jit.profile")
sampletable = {}
jit_p.start("f", function(thread, samples, vmstate)
  local dump=jit_p.dumpstack(thread, "f", 1)
  sampletable[dump] = (sampletable[dump] or 0) + samples
end)
f3()
jit_p.stop()
for d,v in pairs(sampletable) do print(v, d) end
```

Как правило, результат покажет, что выборка многократно осуществлялась в рамках f1(), а также в рамках внутренних функций Tarantool, имена которых могут изменяться с каждой новой версией.

## 6.10 Контроль за фоновыми программами

### 6.10.1 Сигналы от сервера

Во время событийного цикла в потоке обработки транзакций Tarantool обрабатывает следующие сигналы:

| Сигнал | Эффект |
|---|---|
| SIGHUP | Может привести к ротации журналов, см. *пример* в справочнике по параметрам журналирования Tarantool. |
| SIGUSR1 | Может привести к созданию снимка состояния базы данных, см. описание функции *Функция box.snapshot*. |
| SIGTERM | Может привести к корректному завершению работы (с предварительным сохранением всех данных). |
| SIGINT (или «прерывание от клавиатуры») | Может привести к корректному завершению работы. |
| SIGKILL | Приводит к аварийному завершению работы. |

Остальные сигналы приводят к заданному операционной системой поведению. Все сигналы, за исключением SIGKILL, можно игнорировать, особенно если Tarantool выполняет длительную процедуру и не может вернуться в событийный цикл в потоке обработки транзакций.

### 6.10.2 Автоматическая перезагрузка экземпляра

На платформах, где доступна утилита `systemd`, `systemd` автоматически перезагружает все экземпляры Tarantool при сбое. Чтобы продемонстрировать это, отключим один из экземпляров:

```
$ systemctl status tarantool@my_app|grep PID
Main PID: 5885 (tarantool)
$ tt connect my_app
   • Connecting to the instance...
   • Connected to /var/run/tarantool/my_app.control
/var/run/tarantool/my_app.control> os.exit(-1)
    Connection was closed. Probably instance process isn't running anymore
```

А теперь убедимся, что `systemd` перезапустила его:

```
$ systemctl status tarantool@my_app|grep PID
Main PID: 5914 (tarantool)
```

Additionally, you can find the information about the instance restart in the boot logs:

```
$ journalctl -u tarantool@my_app -n 8
```

### 6.10.3 Создание дампов памяти

Tarantool создает дамп памяти при получении одного из следующих сигналов: SIGSEGV, SIGFPE, SIGABRT или SIGQUIT. При сбое Tarantool дамп создается автоматически.

На платформах, где доступна утилита `systemd`, `coredumpctl` автоматически сохраняет дампы памяти и трассировку стека при аварийном завершении Tarantool-сервера. Вот как включить создание дампов памяти в Unix-системе:

1. Убедитесь, что лимиты для сессии установлены таким образом, чтобы можно было создавать дампы памяти, – выполните команду `ulimit -c unlimited`. Также проверьте «man 5 core» на другие причины, по которым дамп памяти может не создаваться.
2. Создайте директорию для записи дампов памяти и убедитесь, что в эту директорию действительно можно производить запись. На Linux путь до директории задается в параметре ядра, который настраивается через `/proc/sys/kernel/core_pattern`.
3. Убедитесь, что дампы памяти включают трассировку стека. При использовании бинарного дистрибутива Tarantool эта информация включается автоматически. При сборке Tarantool из исходников, если передать CMake флаг `-DCMAKE_BUILD_TYPE=Release`, вы не получите подробной информации.

Для симуляции сбоя можно попытаться выполнить нелегальную команду на работающем экземпляре Tarantool:

```
$ # !!! please never do this on a production system !!!
$ tt connect my_app
   • Connecting to the instance...
   • Connected to /var/run/tarantool/my_app.control
/var/run/tarantool/my_app.control> require('ffi').cast('char *', 0)[0] = 48
    Connection was closed. Probably instance process isn't running anymore
```

Есть другой способ: если вы знаете PID экземпляра ($PID в нашем примере), можно остановить этот экземпляр, запустив отладчик `gdb`:

```
$ gdb -batch -ex "generate-core-file" -p $PID
```

или послав вручную сигнал SIGABRT:

```
$ kill -SIGABRT $PID
```

---

**Примечание:** Чтобы узнать PID экземпляра, можно:

- посмотреть его с помощью *box.info.pid*,
- использовать команду `ps -A | grep tarantool`, или
- выполнить `systemctl status tarantool@my_app|grep PID`.

---

Чтобы посмотреть на последние сбои Tarantool-демона на платформах, где доступна утилита `systemd`, выполните команду:

```
$ coredumpctl list /usr/bin/tarantool
MTIME                            PID   UID   GID SIG PRESENT EXE
Sat 2016-01-23 15:21:24 MSK   20681  1000  1000   6   /usr/bin/tarantool
Sat 2016-01-23 15:51:56 MSK   21035   995   992   6   /usr/bin/tarantool
```

Чтобы сохранить дамп памяти в файл, выполните команду:

```
$ coredumpctl -o filename.core info <pid>
```

### 6.10.4 Трассировка стека

Так как Tarantool хранит кортежи в памяти, файлы с дампами памяти могут быть довольно большими. Чтобы найти проблему, обычно целый файл не нужен – достаточно только «трассировки стека» или «обратной трассировки».

Чтобы сохранить трассировку стека в файл, выполните команду:

```
$ gdb -se "tarantool" -ex "bt full" -ex "thread apply all bt" --batch -c core> /tmp/
↪tarantool_trace.txt
```

где:

- «tarantool» – это путь до исполняемого файла Tarantool,
- «core» – это путь до файла с дампом памяти, и
- «/tmp/tarantool_trace.txt» – это пример пути до файла, в который сохраняется трассировка стека.

---

**Примечание:** Иногда может оказаться, что файл с трассировкой стека не содержит отладочных символов – в таких строках вместо имени будет стоять "??". Если это произошло, ознакомьтесь с инструкциями на этих двух wiki-страницах Tarantool: How to debug core dump of stripped tarantool и How to debug core from different OS.

---

Чтобы получить трассировку стека и прочую полезную информацию в консоли, выполните команду:

```
$ coredumpctl info 21035
          PID: 21035 (tarantool)
          UID: 995 (tarantool)
          GID: 992 (tarantool)
       Signal: 6 (ABRT)
    Timestamp: Sat 2016-01-23 15:51:42 MSK (4h 36min ago)
 Command Line: tarantool my_app.lua <running>
   Executable: /usr/bin/tarantool
Control Group: /system.slice/system-tarantool.slice/tarantool@my_app.service
         Unit: tarantool@my_app.service
        Slice: system-tarantool.slice
      Boot ID: 7c686e2ef4dc4e3ea59122757e3067e2
   Machine ID: a4a878729c654c7093dc6693f6a8e5ee
     Hostname: localhost.localdomain
      Message: Process 21035 (tarantool) of user 995 dumped core.

               Stack trace of thread 21035:
               #0  0x00007f84993aa618 raise (libc.so.6)
               #1  0x00007f84993ac21a abort (libc.so.6)
               #2  0x0000560d0a9e9233 _ZL12sig_fatal_cbi (tarantool)
               #3  0x00007f849a211220 __restore_rt (libpthread.so.0)
               #4  0x0000560d0aaa5d9d lj_cconv_ct_ct (tarantool)
               #5  0x0000560d0aaa687f lj_cconv_ct_tv (tarantool)
               #6  0x0000560d0aaabe33 lj_cf_ffi_meta___newindex (tarantool)
               #7  0x0000560d0aaae2f7 lj_BC_FUNCC (tarantool)
               #8  0x0000560d0aa9aabd lua_pcall (tarantool)
               #9  0x0000560d0aa71400 lbox_call (tarantool)
               #10 0x0000560d0aa6ce36 lua_fiber_run_f (tarantool)
               #11 0x0000560d0a9e8d0c _ZL16fiber_cxx_invokePFiP13__va_list_tagES0_␣
↪(tarantool)
               #12 0x0000560d0aa7b255 fiber_loop (tarantool)
               #13 0x0000560d0ab38ed1 coro_init (tarantool)
               ...
```

### 6.10.5 Отладчик

Для запуска отладчика gdb, выполните команду:

```
$ coredumpctl gdb <pid>
```

Мы очень рекомендуем установить пакет tarantool-debuginfo, чтобы сделать отладку средствами gdb более эффективной. Например:

```
$ dnf debuginfo-install tarantool
```

С помощью gdb можно узнать, какие еще debuginfo-пакеты нужно установить:

```
$ gdb -p <pid>
...
Missing separate debuginfos, use: dnf debuginfo-install
glibc-2.22.90-26.fc24.x86_64 krb5-libs-1.14-12.fc24.x86_64
libgcc-5.3.1-3.fc24.x86_64 libgomp-5.3.1-3.fc24.x86_64
```

(continues on next page)

(продолжение с предыдущей страницы)

```
libselinux-2.4-6.fc24.x86_64 libstdc++-5.3.1-3.fc24.x86_64
libyaml-0.1.6-7.fc23.x86_64 ncurses-libs-6.0-1.20150810.fc24.x86_64
openssl-libs-1.0.2e-3.fc24.x86_64
```

В трассировке стека присутствуют символические имена, даже если у вас не установлен пакет `tarantool-debuginfo`.

## 6.11 Аварийное восстановление

The minimal fault-tolerant Tarantool configuration would be a *replica set* that includes a master and a replica, or two masters. The basic recommendation is to configure all Tarantool instances in a replica set to create *snapshot files* on a regular basis.

Here are action plans for typical crash scenarios.

### 6.11.1 Мастер-реплика

#### Master crash: manual failover

**Configuration:** master-replica (*manual failover*).

**Problem:** The master has crashed.

**Actions:**

1. Ensure the master is stopped. For example, log in to the master machine and use `tt stop`.
2. Configure a new replica set leader using the *<replicaset_name>.leader* option.
3. Reload configuration on all instances using *config:reload()*.
4. Make sure that a new replica set leader is a master using *box.info.ro*.
5. On a new master, *remove a crashed instance from the „_cluster“ space*.
6. Set up a replacement for the crashed master on a spare host.

See also: *Performing manual failover*.

#### Master crash: automated failover

**Configuration:** master-replica (*automated failover*).

**Problem:** The master has crashed.

**Actions:**

1. Use `box.info.election` to make sure a new master is elected automatically.
2. On a new master, *remove a crashed instance from the „_cluster“ space*.
3. Set up a replacement for the crashed master on a spare host.

See also: *Testing automated failover*.

#### Потеря данных

**Configuration:** master-replica.

**Problem:** Some transactions are missing on a replica after the master has crashed.

**Actions:**

You lose a few transactions in the master *write-ahead log file*, which may have not transferred to the replica before the crash. If you were able to salvage the master `.xlog` file, you may be able to recover these.

1. Посмотрите UUID экземпляра в *xlog-файле* вышедшего из строя мастера:

```
$ head -5 var/lib/instance001/*.xlog | grep Instance
Instance: 9bb111c2-3ff5-36a7-00f4-2b9a573ea660
```

2. Используйте этот UUID на новом мастере для поиска позиции:

```
app:instance002> box.info.vclock[box.space._cluster.index.uuid:select{'9bb111c2-
↪3ff5-36a7-00f4-2b9a573ea660'}[1][1]]
---
- 999
...
```

3. *Play the records* from the crashed `.xlog` to the new master, starting from the new master position:

```
$ tt play 127.0.0.1:3302 var/lib/instance001/00000000000000000000.xlog \
          --from 1000 \
          --replica 1 \
          --username admin --password secret
```

### 6.11.2 Мастер-мастер

**Configuration:** *master-master*.

**Problem:** one master has crashed.

**Actions:**

1. Let the load be handled by another master alone.
2. Remove a crashed master from a replica set.
3. Set up a replacement for the crashed master on a spare host. Learn more from *Adding and removing instances*.

### 6.11.3 Master-replica/master-master: data loss

**Configuration:** master-replica or master-master.

**Problem:** Data was deleted at one master and this data loss was propagated to the other node (master or replica).

**Actions:**

1. Put all nodes in read-only mode. Depending on the *replication.failover* mode, this can be done as follows:
   - `manual`: change a replica set leader to `null`.

- `election`: set *replication.election_mode* to `voter` or `off` at the replica set level.
- `off`: set `database.mode` to `ro`.

Reload configurations on all instances using the `reload()` function provided by the *config* module.

2. Turn off deletion of expired checkpoints with *box.backup.start()*. This prevents the Tarantool garbage collector from removing files made with older checkpoints until *box.backup.stop()* is called.
3. Get the latest valid *.snap file* and use `tt cat` command to calculate at which LSN the data loss occurred.
4. Start a new instance and use *tt play* command to play to it the contents of `.snap` and `.xlog` files up to the calculated LSN.
5. Bootstrap a new replica from the recovered master.

---

**Примечание:** The steps above are applicable only to data in the memtx storage engine.

---

## 6.12 Резервное копирование

Архитектура Tarantool-хранилища позволяет производить обновление только путем присоединения новых записей: сами файлы никогда не перезаписываются. *Сборщик мусора Tarantool* удаляет старые файлы после определенной контрольной точки. В настройках *демона создания контрольных точек* можно отложить или запретить работу сборщика мусора. Резервное копирование может проводиться в любое время с минимальной затратой ресурсов.

Для резервного копирования в определенных ситуациях используются две функции:

- *box.backup.start()* сообщает серверу, что следует отложить некоторые действия, связанные с удалением устаревших резервных копий, и возвращает таблицу с именами файлов снимков и файлов vinyl'а, которые необходимо будет скопировать.
- *box.backup.stop()* затем уведомляет сервер, что работа может быть продолжена в обычном режиме.

### 6.12.1 Горячее резервное копирование (memtx)

Это особый случай, когда все таблицы хранятся в памяти.

Последний созданный Tarantool *файл-снимок* является резервной копией всей базы данных; а *WAL-файлы*, созданные следом за последним файлом-снимком, являются инкрементными копиями. Поэтому процедура резервного копирования сводится к копированию последнего файла-снимка и следующих за ним WAL-файлов.

1. Use `tar` to make a (possibly compressed) copy of the latest .snap and .xlog files on the *snapshot.dir* and *wal.dir* directories.
2. Если того требуют правила безопасности, зашифруйте получившийся .tar-файл.
3. Скопируйте .tar-файл в надежное место.

Later, restoring the database is a matter of taking the .tar file and putting its contents back in the `snapshot.dir` and `wal.dir` directories.

### 6.12.2 Горячее резервное копирование (vinyl/memtx)

Vinyl хранит свои файлы в *vinyl_dir* и создает для каждого спейса в базе данных отдельную поддиректорию. Создание дампов и слияние – это процессы, которые могут лишь добавлять записи, поэтому в результате создаются новые файлы. Сборщик мусора Tarantool может удалять старые файлы после каждой контрольной точки.

Для создания смешанной резервной копии:

1. Выполните команду *box.backup.start()* в *административной консоли*. Эта команда покажет список файлов для резервного копирования и приостановит сборку мусора до следующего вызова `box.backup.stop()`.
2. Скопируйте файлы из списка в надежное место. Это касается файлов-снимков memtx, выполняемых vinyl-файлов и индексных файлов, соответствующих последней контрольной точке.
3. Выполните команду *box.backup.stop()*, чтобы сборщик мусора мог продолжить работу.

### 6.12.3 Непрерывное удаленное резервное копирование

*Репликация* обеспечивает резервное копирование и помогает балансировать нагрузку.

Поэтому процесс создания резервной копии сводится к обновлению (при необходимости) одной из реплик с последующим холодным резервным копированием. Так как все остальные реплики продолжают функционировать, с точки зрения конечного пользователя, этот процесс не является холодным резервным копированием. Такое копирование можно выполнять регулярно с помощью планировщика `cron` или файбера Tarantool.

### 6.12.4 Непрерывное резервное копирование

По ходу работы системы необходимо сохранять записи об изменениях, внесенных со времени последнего холодного резервного копирования.

Для этого нужна специальная утилита для копирования файлов (например, rsync), которая позволит удаленно и на постоянной основе копировать только изменившиеся части WAL-файла, а не весь файл целиком.

Можно взять и обычную утилиту для копирования целых файлов, но тогда придется создавать файлы-снимки и WAL-файлы на каждое изменение, чтобы нужно было копировать только новые файлы.

## 6.13 Обновление

This section describes the general upgrade process for Tarantool. There are two main upgrade scenarios for different use cases:

- *Live upgrade* (without downtime) for replication clusters.
- *Upgrade with downtime* for standalone instances.

You can also downgrade to an earlier version using a similar procedure.

For information about backwards compatibility, see the *compatibility guarantees* description.

Upgrading from or to certain versions can involve specific steps or slightly differ from the general upgrade procedure. Such version-specific cases are described on the dedicated pages inside this section.

This section includes the following topics:

### 6.13.1 Standalone instance upgrade

This page describes the process of upgrading a standalone Tarantool instance in production. Note that this **always implies a downtime** because the application needs to be stopped and restarted on the target version.

To upgrade **without downtime**, you need multiple Tarantool servers running in a replication cluster. Find detailed instructions in *Replication cluster upgrade*.

#### Checking your application

Before upgrading, make sure your application is compatible with the target Tarantool version:

1. Set up a development environment with the target Tarantool version installed. See the installation instructions at the Tarantool download page and in the *tt install reference*.
2. Deploy the application in this environment and check how it works. In case of any issues, adjust the application code to ensure compatibility with the target version.

When your application is ready to run on the target Tarantool version, you can start upgrading the production environment.

#### Upgrading a standalone instance

1. Stop the Tarantool instance.
2. Make a copy of all data and the package from which the current (old) version was installed. You may need it for rollback purposes. Find the backup instruction in the appropriate hot backup procedure in *Backups*.
3. Install the target Tarantool version on the host. You can do this using a package manager or the *tt utility*. See the installation instructions at Tarantool download page and in the *tt install reference*. To check that the target Tarantool version is installed, run `tarantool -v`.
4. Start your application on the target version.
5. Run *box.schema.upgrade()*. This will update the Tarantool system spaces to match the currently installed version of Tarantool.

   ---

   **Примечание:** To undo schema upgrade in a case of failed upgrade, you can use *box.schema.downgrade()*.

   ---

#### Rollback

The rollback procedure for a standalone instance is almost the same as the upgrade. The only difference is in the last step: you should call *box.schema.downgrade()* to return the schema to the original version.

### 6.13.2 Replication cluster upgrade

Below are the general instructions for upgrading a Tarantool cluster with replication. Upgrading from some versions can involve certain specifics. To find out if it is your case, check the version-specific topics of the *Upgrades* section.

A replication cluster can be upgraded without downtime due to its redundancy. When you disconnect a single instance for an upgrade, there is always another instance that takes over its functionality: being a master storage for the same data buckets or working as a router. This way, you can upgrade all the instances one by one.

The high-level steps of cluster upgrade are the following:

1. *Ensure the application compatibility* with the target Tarantool version.
2. *Check the cluster health*.
3. *Install the target Tarantool version* on the cluster nodes.
4. *Upgrade router* nodes one by one.
5. *Upgrade storage replica sets* one by one.

---

**Важно:** The only way to upgrade Tarantool from version 1.6, 1.7, or 1.9 to 2.x **without downtime** is to take an intermediate step by upgrading to 1.10 and then to 2.x.

Before upgrading Tarantool from 1.6 to 2.x, please read about the associated *caveats*.

---

---

**Примечание:** Some upgrade steps are moved to the separate section *Procedures and checks* to avoid overloading the general instruction with details. Typically, these are checks you should repeat during the upgrade to ensure it goes well.

---

If you experience issues during upgrade, you can roll back to the original version. The rollback instructions are provided in the *Rollback* section.

#### Checking your application

Before upgrading, make sure your application is compatible with the target Tarantool version:

1. Set up a development environment with the target Tarantool version installed. See the installation instructions at the Tarantool download page and in the *tt install reference*.
2. Deploy the application in this environment and check how it works. In case of any issues, adjust the application code to ensure compatibility with the target version.

When your application is ready to run on the target Tarantool version, you can start upgrading the production environment.

### Pre-upgrade checks

Perform these steps before the upgrade to ensure that your cluster is working correctly:

1. On each `router` instance, perform the *vshard.router check*:

```
tarantool> vshard.router.info()
-- no issues in the output
-- sum of 'bucket.available_rw' == total number of buckets
```

2. On each `storage` instance, perform the *replication check*:

```
tarantool> box.info
-- box.info.status == 'running'
-- box.info.ro == 'false' on one instance in each replica set.
-- box.info.replication[*].upstream.status == 'follow'
-- box.info.replication[*].downstream.status == 'follow'
-- box.info.replication[*].upstream.lag <= box.cfg.replication_timeout
-- can also be moderately larger under a write load
```

3. On each `storage` instance, perform the *vshard.storage check*:

```
tarantool> vshard.storage.info()
-- no issues in the output
-- replication.status == 'follow'
```

4. Check all instances" *logs* for application errors.

---

**Примечание:** If you're running Cartridge, you can check the health of the cluster instances on the **Cluster** tab of its web interface.

---

In case of any issues, make sure to fix them before starting the upgrade procedure.

### Installing the target version

Install the target Tarantool version on all hosts of the cluster. You can do this using a package manager or the *tt utility*. See the installation instructions at the Tarantool download page and in the *tt install reference*.

Check that the target Tarantool version is installed by running `tarantool -v` on all hosts.

### Upgrading a Tarantool cluster with no downtime

#### Upgrading routers

Upgrade **router** instances one by one:

1. Stop one `router` instance.
2. Start this instance on the target Tarantool version.
3. Repeat the previous steps for each `router` instance.

After completing the router instances upgrade, perform the *vshard.router check* on each of them.

### Upgrading storages

Before upgrading **storage** instances:

- Disable Cartridge failover: run

```
tt cartridge failover disable
```

  or use the Cartridge web interface (**Cluster** tab, **Failover:** <**Mode**> button).

- Disable the *rebalancer*: run

```
tarantool> vshard.storage.rebalancer_disable()
```

- Make sure that the Cartridge `upgrade_schema` option is `false`.

Upgrade storage instances by performing the following steps for each replica set:

---

**Примечание:** To detect possible upgrade issues early, we recommend that you perform a *replication check* on all instances of the replica set **after each step**.

---

1. Pick a replica (a **read-only** instance) from the replica set. Stop this replica and start it again on the target Tarantool version. Wait until it reaches the `running` status (`box.info.status == running`).
2. Restart all other **read-only** instances of the replica set on the target version one by one.
3. Make one of the updated replicas the new master using the applicable instruction from *Switching the master*.
4. Restart the last instance of the replica set (the former master, now a replica) on the target version.
5. Run *box.schema.upgrade()* on the new master. This will update the Tarantool system spaces to match the currently installed version of Tarantool. The changes will be propagated to other nodes via the replication mechanism later.

**Предупреждение:** This is the point of no return for upgrading from versions earlier than 2.8.2: once you complete it, the schema is no longer compatible with the initial version.

When upgrading from version 2.8.2 or newer, you can undo the schema upgrade using *box.schema.downgrade()*.

6. Run `box.snapshot()` on every node in the replica set to make sure that the replicas immediately see the upgraded database state in case of restart.

Once you complete the steps, enable failover or rebalancer back:

- Enable Cartridge failover: run

```
tt cartridge failover set [mode]
```

  or use the Cartridge web interface (**Cluster** tab, **Failover: Disabled** button).

- Enable the *rebalancer*: run

```
tarantool> vshard.storage.rebalancer_enable()
```

### Post-upgrade checks

Perform these steps after the upgrade to ensure that your cluster is working correctly:

1. On each `router` instance, perform the *vshard.router check*:

```
tarantool> vshard.router.info()
-- no issues in the output
-- sum of 'bucket.available_rw' == total number of buckets
```

2. On each `storage` instance, perform the *replication check*:

```
tarantool> box.info
-- box.info.status == 'running'
-- box.info.ro == 'false' on one instance in each replica set.
-- box.info.replication[*].upstream.status == 'follow'
-- box.info.replication[*].downstream.status == 'follow'
-- box.info.replication[*].upstream.lag <= box.cfg.replication_timeout
-- can also be moderately larger under a write load
```

3. On each `storage` instance, perform the *vshard.storage check*:

```
tarantool> vshard.storage.info()
-- no issues in the output
-- replication.status == 'follow'
```

4. Check all instances" *logs* for application errors.

---

**Примечание:** If you're running Cartridge, you can check the health of the cluster instances on the **Cluster** tab of its web interface.

---

### Rollback

#### Rollback before the point of no return

If you decide to roll back before reaching the *point of no return*, your data is fully compatible with the version you had before the upgrade. In this case, you can roll back the same way: restart the nodes you've already upgraded on the original version.

#### Rollback after the point of no return

If you've passed the *point of no return* (that is, executed `box.schema.upgrade()`) during the upgrade, then a rollback requires downgrading the schema to the original version.

To check if an automatic downgrade is available for your original version, use `box.schema.downgrade_versions()`. If the version you need is on the list, execute the following steps on **each upgraded replica set** to roll back:

1. Run `box.schema.downgrade(<version>)` on master specifying the original version.
2. Run `box.snapshot()` on every instance in the replica set to make sure that the replicas immediately see the downgraded database state after restart.
3. Restart all **read-only** instances of the replica set on the initial version one by one.

4. Make one of the updated replicas the new master using the applicable instruction from *Switching the master*.
5. Restart the last instance of the replica set (the former master, now a replica) on the original version.

Then enable failover or rebalancer back as described in the *Upgrading storages*.

### Recovering from a failed upgrade

**Предупреждение:** This section applies to cases when the upgrade procedure has failed and the cluster is not functioning properly anymore. Thus, it implies a downtime and a full cluster restart.

In case of an upgrade failure after passing the *point of no return*, follow these steps to roll back to the original version:

1. Stop all cluster instances.
2. Save snapshot and `xlog` files from all instances whose data was modified after the last backup procedure. These files will help apply these modifications later.
3. Save the latest backups from all instances.
4. Restore the original Tarantool version on all hosts of the cluster.
5. Launch the cluster on the original Tarantool version.

   **Примечание:** At this point, the application becomes fully functional and contains data from the backups. However, the data modifications made after the backups were taken must be restored manually.

6. Manually apply the latest data modifications from `xlog` files you saved on step 2 using the *xlog* module. On instances where such changes happened, do the following:
   1. Find out the vclock value of the latest operation in the original WAL.
   2. Play the operations from the newer xlog starting from this vclock on the instance.

   **Важно:** If the upgrade has failed after calling `box.schema.upgrade()`, **don't apply** the modifications of system spaces done by this call. This can make the schema incompatible with the original Tarantool version.

Find more information about the Tarantool recovery in *Disaster recovery*.

### Procedures and checks

#### Replication check

Run `box.info`:

```
tarantool> box.info
```

Check that the following conditions are satisfied:

- `box.info.status` is `running`

- `box.info.replication[*].upstream.status` and `box.info.replication[*].downstream.status` are `follow`
- `box.info.replication[*].upstream.lag` is less or equal than `box.cfg.replication_timeout`, but it can also be moderately larger under a write load.
- `box.info.ro` is `false` at least on one instance in **each** replica set. If all instances have `box.info.ro = true`, this means there are no writable nodes. On Tarantool *v. 2.10.0* or later, you can find out why this happened by running `box.info.ro_reason`. If `box.info.ro_reason` or `box.info.status` has the value `orphan`, the instance doesn't see the rest of the replica set.

Then run `box.info` once more and check that `box.info.replication[*].upstream.lag` values are updated.

#### vshard.storage check

Run `vshard.storage.info()`:

```
tarantool> vshard.storage.info()
```

Check that the following conditions are satisfied:

- there are no issues or alerts
- `replication.status` is `follow`

#### vshard.router check

Run `vshard.router.info()`:

```
tarantool> vshard.router.info()
```

Check that the following conditions are satisfied:

- there are no issues or alerts
- all buckets are available (the sum of `bucket.available_rw` on all replica sets equals the total number of buckets)

### Switching the master

- **Cartridge**. If your cluster runs on Cartridge, you can switch the master in the web interface. To do this, go to the **Cluster** tab, click **Edit replica set**, and drag an instance to the top of **Failover priority** list to make it the master.
- **Raft**. If your cluster uses *automated leader election*, switch the master by following these steps:
  1. Pick a *candidate* – a read-only instance to become the new master.
  2. Run `box.ctl.promote()` on the candidate. The operation will start and wait for the election to happen.
  3. Run `box.cfg{ election_mode = "voter" }` on the current master.
  4. Check that the candidate became the new master: its `box.info.ro` must be `false`.
- **Legacy**. If your cluster neither works on Cartridge nor has automated leader election, switch the master by following these steps:

1. Pick a *candidate* – a read-only instance to become the new master.
2. Run `box.cfg{ read_only = true }` on the current master.
3. Check that the candidate's vclock value matches the master's: The value of `box.info.vclock[<master_id>]` on the candidate must be equal to `box.info.lsn` on the master. `<master_id>` here is the value of `box.info.id` on the master.

   If the vclock values don't match, stop the switch procedure and restore the replica set state by calling `box.cfg{ read_only == false }` on the master. Then pick another candidate and restart the procedure.

After switching the master, perform the *replication check* on each instance of the replica set.

### 6.13.3 Live upgrade from Tarantool 1.6 to 1.10

This page includes explanations and solutions to some common issues when upgrading a replica set from Tarantool 1.6 to 1.10.

Versions later that 1.6 have incompatible *.snap* and *.xlog* file formats: 1.6 files are supported during upgrade, but you won't be able to return to 1.6 after running under 1.10 or 2.x for a while. A few configuration parameters are also renamed.

To perform a **live** upgrade from Tarantool 1.6 to a more recent version, like *2.8.4*, *2.10.1* and such, it is necessary to take an intermediate step by upgrading 1.6 -> **1.10** -> 2.x. This is the only way to perform the upgrade **without downtime**.

However, a direct upgrade of a replica set from 1.6 to 2.x is also possible, but only **with downtime**.

The procedure of live upgrade from 1.6 to 1.10 is similar to the general *cluster upgrade procedure*, but with slight differences in the *Upgrading storages* step. Find below the general storage upgrade procedure and the 1.6-specific notes for its steps.

#### General storage upgrade

Upgrade storage instances by performing the following steps for each replica set:

---

**Примечание:** To detect possible upgrade issues early, we recommend that you perform a *replication check* on all instances of the replica set **after each step**.

---

1. Pick a replica (a **read-only** instance) from the replica set. Stop this replica and start it again on the target Tarantool version. Wait until it reaches the `running` status (`box.info.status == running`).
2. Restart all other **read-only** instances of the replica set on the target version one by one.
3. Make one of the updated replicas the new master using the applicable instruction from *Switching the master*.
4. Restart the last instance of the replica set (the former master, now a replica) on the target version.
5. Run *box.schema.upgrade()* on the new master. This will update the Tarantool system spaces to match the currently installed version of Tarantool. The changes will be propagated to other nodes via the replication mechanism later.
6. Run `box.snapshot()` on every node in the replica set to make sure that the replicas immediately see the upgraded database state in case of restart.

#### 1.6 storage upgrade specifics

- **Replication check:** New Tarantool nodes follow 1.6 nodes just fine, but some 1.6 nodes might disconnect from new nodes with an ER_LOADING error. This is not critical, the error goes away when replication on 1.6 is restarted:

```
old_repl = box.cfg.replication
box.cfg{replication = ""}
box.cfg{replication = old_repl}
```

- **Point of no return:** When upgrading from Tarantool 1.6, the step 3 (switching the master) is the point of no return. Once you complete it, **the schema is no longer compatible with the initial version**.
- **Restarting on the target version (steps 1, 2, and 4):** Tarantool 1.10+ fails to recover from 1.6 xlogs, unless `box.cfg{force_recovery = true}` is set. There is a slight difference between 1.6 and 1.10 xlogs, which makes 1.6 xlogs appear erroneous to 1.10+ instances. In order to work around this, start the instance in `force_recovery` mode. To do so, add the line `force_recovery = true` to the file where the instance is initialized – for example, to `init.lua`.
- **Running box.schema.upgrade() (step 5):** There was a breaking change between 1.6 and 1.10 – in 1.6, the field type `num` was an alias to `number`, and in 1.10, `num` is converted to `unsigned`. This means that after `box.schema.upgrade()` is performed on the master, the user might have some spaces with `unsigned` fields containing non-unsigned values: `double`, `int`, and so on. This will make the snapshot inconsistent, unless an extra action is performed after `box.schema.upgrade()`. Run this code in the Tarantool console on the new master:

```
-- First find all spaces containing unsigned fields with non-unsigned values in␣
↪them.
-- Say, we have one such space denoted problematic_space and the problem is in␣
↪field problematic_field_no.
a = box.space.problematic_space:format()
a[problematic_field_no].type = 'number'
box.space.problematic_space:format(a)
```

- **Taking snapshots (step 6):** The user might be concerned with snapshot size in 1.10 – it's drastically smaller than the one created by 1.6 (for example, ~300 Mb vs. 6 Gb in some corner cases). There is nothing to worry about. Tarantool 1.6 didn't compress snapshots, while Tarantool 1.10 and above does that.

### 6.13.4 Upgrade from 1.6 directly to 2.x with downtime

Versions later that 1.6 have incompatible *.snap* and *.xlog* file formats: 1.6 files are supported during upgrade, but you won't be able to return to 1.6 after running under 1.10 or 2.x for a while. A few configuration parameters are also renamed.

To perform a **live** upgrade from Tarantool 1.6 to a more recent version, like *2.8.4*, *2.10.1* and such, it is necessary to take an intermediate step by upgrading 1.6 -> **1.10** -> 2.x. This is the only way to perform the upgrade **without downtime**.

However, a direct upgrade of a replica set from 1.6 to 2.x is also possible, but only **with downtime**.

Here is how to upgrade from Tarantool 1.6 directly to 2.x:

1. Stop all instances in the replica set.
2. *Upgrade* Tarantool version to 2.x on every instance.

3. Upgrade the corresponding instance files and applications, if needed.
4. Start all the instances with Tarantool 2.x.
5. Execute `box.schema.upgrade()` on the master.
6. Execute `box.snapshot()` on every node in the replica set.

### 6.13.5 Fix decimal values in vinyl spaces when upgrading to 2.10.1

This is an upgrade guide for fixing one specific problem which could happen with decimal values in vinyl spaces. It's only relevant when you're upgrading from Tarantool version <= 2.10.0 to anything >= 2.10.1.

Before gh-6377 was fixed, `decimal` and `double` values in a `scalar` or `number` index could end up in the wrong order after the update. If such an index has been built for a space that uses the `vinyl` storage engine, the index is persisted and is not rebuilt even after the upgrade. If this is the case, the user has to rebuild the affected indexes manually.

Here are the rules to determine whether your installation was affected. If all of the statements listed below are true, you have to rebuild indexes for the affected `vinyl` spaces manually.

- You were running Tarantool version 2.10.0 and below.
- You have spaces with the `vinyl` storage engine.
- The `vinyl` spaces have `number` or `scalar` indexes.
- The tuples in these spaces may contain both `decimal` and `double Inf` or `NaN` values.

If this is the case for you, you can run the following script, which will find all the affected indices:

```
local fiber = require('fiber')
local decimal = require('decimal')

local function isnan(val)
    return type(val) == 'number' and val ~= val
end

local function isinf(val)
    return val == math.huge or val == -math.huge
end

local function vinyl(id)
    return box.space[id].engine == 'vinyl'
end

require_rebuild = {}
local iters = 0
for _, v in box.space._index:pairs({512, 0}, {iterator='GE'}) do
    local id = v[1]
    iters = iters + 1
    if iters % 1000 == 0 then
        fiber.yield()
    end
    if vinyl(id) then
        local format = v[6]
        local check_fields = {}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        for _, fmt in pairs(v[6]) do
            if fmt[2] == 'number' or fmt[2] == 'scalar' then
                table.insert(check_fields,  fmt[1] + 1)
            end
        end
        local have_decimal = {}
        local have_nan = {}
        if #check_fields > 0 then
            for k, tuple in box.space[id]:pairs() do
                for _, i in pairs(check_fields) do
                    iters = iters + 1
                    if iters % 1000 == 0 then
                        fiber.yield()
                    end
                    have_decimal[i] = have_decimal[i] or
                                      decimal.is_decimal(tuple[i])
                    have_nan[i] = have_nan[i] or isnan(tuple[i]) or
                                  isinf(tuple[i])
                    if have_decimal[i] and have_nan[i] then
                        table.insert(require_rebuild, v)
                        goto out
                    end
                end
            end
        end
    end
    ::out::
end
```

The indices requiring a rebuild will be stored in the `require_rebuild` table. If the table is empty, you're safe and can continue using Tarantool as before.

If the `require_rebuild` table contains some entries, you can rebuild the affected indices with the following script.

---

**Примечание:** Please run the script below only on the master node and only after all the nodes are upgraded to the new Tarantool version.

---

```
local log = require('log')

local function rebuild_index(idx)
    local index_name = idx[3]
    local space_name = box.space[idx[1]].name
    log.info("Rebuilding index %s on space %s", index_name, space_name)
    if (idx[2] == 0) then
        log.error("Cannot rebuild primary index %s on space %s. Please, "..
                "recreate the space manually", index_name, space_name)
        return
    end
    log.info("Deleting index %s on space %s", index_name, space_name)
    local v = box.space._index:delete{idx[1], idx[2]}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    if v == nil then
        log.error("Couldn't find index %s on space %s", index_name, space_name)
        return
    end
    log.info("Done")
    log.info("Creating index %s on space %s", index_name, space_name)
    box.space._index:insert(v)
end

for _, idx in pairs(require_rebuild) do
    rebuild_index(idx)
end
```

The script might fail on some of the indices with the following error: «Cannot rebuild primary index index_name on space space_name. Please, recreate the space manually». If this happens, automatic index rebuild is impossible, and you have to manually re-create the space to ensure data integrity:

1. Create a new space with the same format as the existing one.
2. Define the same indices on the freshly created space.
3. Iterate over the old space's primary key and insert all the data into the new space.
4. Drop the old space.

### 6.13.6 Fix illegal type names when upgrading to 2.10.4

This is an upgrade guide for fixing one specific problem which could happen with field type names. It's only relevant when you're upgrading from a Tarantool version <=2.10.3 to >=2.10.4.

Before gh-5940 was fixed, the empty string, `n`, `nu`, `s`, and `st` (that is, leading parts of `num` and `str`) were accepted as valid field types. Since 2.10.4, Tarantool doesn't accept these strings and they must be replaced with correct values `num` and `str`.

This instruction is also available on GitHub.

#### Check if your snapshots contain illegal type names

A snapshot can be validated against the issue using the following script:

```
#!/usr/bin/env tarantool

local xlog = require('xlog')
local json = require('json')

if arg[1] == nil then
    print(('Usage: %s xxxxxxxxxxxxxxxxxxxx.snap'):format(arg[0]))
    os.exit(1)
end

local illegal_types = {
    [''] = true,
    ['n'] = true,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    ['nu'] = true,
    ['s'] = true,
    ['st'] = true,
}

local function report_field_def(name, field_def)
    local msg = 'A field def in a _space entry %q contains an illegal type: %s'
    print(msg:format(name, json.encode(field_def)))
end

local has_broken_format = false

for _, record in xlog.pairs(arg[1]) do
    -- Filter inserts.
    if record.HEADER == nil or record.HEADER.type ~= 'INSERT' then
        goto continue
    end
    -- Filter _space records.
    if record.BODY == nil or record.BODY.space_id ~= 280 then
        goto continue
    end

    local tuple = record.BODY.tuple
    local name = tuple[3]
    local format = tuple[7]

    local is_format_broken = false
    for _, field_def in ipairs(format) do
        if illegal_types[field_def.type] ~= nil then
            report_field_def(name, field_def)
            is_format_broken = true
        end

        if illegal_types[field_def[2]] ~= nil then
            report_field_def(name, field_def)
            is_format_broken = true
        end

    end

    if is_format_broken then
        has_broken_format = true
        local msg = 'The following _space entry contains illegal type(s): %s'
        print(msg:format(json.encode(record)))
    end
    ::continue::
end

if has_broken_format then
    print('')
    print(('%s has an illegal type in a space format'):format(arg[1]))
    print('It is recommended to proceed with the upgrade instruction:')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    print('https://github.com/tarantool/tarantool/wiki/Fix-illegal-field-type-in-a-space-
↪format-when-upgrading-to-2.10.4')
else
    print('Everything looks nice!')
end

os.exit(has_broken_format and 2 or 0)
```

If the snapshot contains the values that aren't valid in 2.10.4, you'll get an output like the following:

### Fix an application file

To fix the application file that contains illegal type names, add the following code in it **before** the `box.cfg()`/`vshard.cfg()`/`cartridge.cfg()` call.

**Примечание:** In Cartridge applications, the instance file is called `init.lua`.

```
-- Convert illegal type names in a space format that were
-- allowed before tarantool 2.10.4.

local log = require('log')
local json = require('json')

local transforms = {
    [''] = 'num',
    ['n'] = 'num',
    ['nu'] = 'num',
    ['s'] = 'str',
    ['st'] = 'str',
}

-- The helper for before_replace().
local function transform_field_def(name, field_def, field, new_type)
    local field_def_old_str = json.encode(field_def)
    field_def[field] = new_type
    local field_def_new_str = json.encode(field_def)

    local msg = 'Transform a field def in a _space entry %q: %s -> %s'
    log.info(msg:format(name, field_def_old_str, field_def_new_str))
end

-- _space trigger.
local function before_replace(_, tuple)
    if tuple == nil then return tuple end

    local name = tuple[3]
    local format = tuple[7]

    -- Update format if necessary.
    local is_format_changed = false
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    for i, field_def in ipairs(format) do
        local new_type = transforms[field_def.type]
        if new_type ~= nil then
            transform_field_def(name, field_def, 'type', new_type)
            is_format_changed = true
        end

        local new_type = transforms[field_def[2]]
        if new_type ~= nil then
            transform_field_def(name, field_def, 2, new_type)
            is_format_changed = true
        end
    end

    -- No changed: skip.
    if not is_format_changed then return tuple end

    -- Rebuild the tuple.
    local new_tuple = tuple:transform(7, 1, format)
    log.info(('Transformed _space entry %s to %s'):format(
        json.encode(tuple), json.encode(new_tuple)))
    return new_tuple
end

-- on_schema_init trigger to set before_replace().
local function on_schema_init()
    box.space._space:before_replace(before_replace)
end

-- Set the trigger on _space.
box.ctl.on_schema_init(on_schema_init)
```

You can delete these triggers **after** the `box.cfg()`/`vshard.cfg()`/`cartridge.cfg()` call.

An example for a Cartridge application:

The triggers will report the changes the make in the following form:

### 6.13.7 Recover from WALs with mixed transactions when upgrading to 2.11.0

This is a guide on fixing a specific problem that could happen when upgrading from a Tarantool version between 2.1.2 and 2.2.0 to 2.8.1 or later. The described solution is applicable since version 2.11.0.

The problem is described in the issue gh-7932. If two or more transactions happened simultaneously in Tarantool 2.1.2-2.2.0, their operations could be written to the write-ahead log mixed with each other. Starting from version 2.8.1, Tarantool recovers transactions atomically and expects all WAL entries between a transaction's `begin` and `commit` operations to belong to one transaction. If there is an operation belonging to another transaction, Tarantool fails to recover from such a WAL.

Starting from version *2.11.0*, Tarantool can recover from WALs with mixed transactions in the `force_recovery` mode.

#### Instances fail to start

If all instances or some of them fail to start after upgrading to 2.11 or a newer version due to a recovery error:

1. Start these instances with the *force_recovery* option to `true`.
2. Make new snapshots on the instances so that the old WALs with mixed transactions aren't used for recovery anymore. To do this, call *box.snapshot()*.
3. Set `force_recovery` back to `false`.

#### Replication doesn't work

After all the instances start successfully, WALs with mixed transactions may still lead to replication issues. Some instances may fail to replicate from other instances because they are sending incorrect WALs. To fix the replication issues, rebootstrap the instances that fail to replicate.

## 6.14 Замечания по поводу некоторых операционных систем

### 6.14.1 macOS

On macOS, no native system tools for administering Tarantool are supported. The recommended way to administer Tarantool instances is using *tt CLI*.

### 6.14.2 Gentoo Linux

В разделе ниже описывается пакет «dev-db/tarantool», установленный из официального оверлея layman (под названием `tarantool`).

По умолчанию с экземплярами используется директория `/etc/tarantool/instances.available`, ее можно переопределить в `/etc/default/tarantool`.

Управление экземплярами Tarantool (запуск/остановка/перезагрузка/проверка статуса и т.д.) можно осуществлять с помощью OpenRC. Рассмотрим пример, как создать экземпляр с управлением OpenRC:

```
$ cd /etc/init.d
$ ln -s tarantool your_service_name
$ ln -s /usr/share/tarantool/your_service_name.lua /etc/tarantool/instances.available/
↪your_service_name.lua
```

Проверяем, что работает:

```
$ /etc/init.d/your_service_name start
$ tail -f -n 100 /var/log/tarantool/your_service_name.log
```

### 6.14.3 FreeBSD

To learn about specifics of using the deprecated `tarantoolctl` utility on FreeBSD, check its *documentation*.

## 6.15 Сообщения об ошибках

Если вы нашли ошибку в Tarantool, вы окажете нам услугу, сообщив о ней.

Пожалуйста, откройте тикет в репозитории Tarantool на GitHub. Рекомендуем включить следующую информацию:

- Шаги для воспроизведения ошибки с объяснением того, как ошибочное поведение отличается от описанного в документации ожидаемого поведения. Пожалуйста, указывайте как можно более конкретную информацию. Например, вместо «Я не могу получить определенную информацию» лучше написать «box.space.x:delete() не указывает, что именно было удалено».
- Название и версию вашей операционной системы, название и версию Tarantool и любую информацию об особенностях вашей машины и ее конфигурации.
- Сопутствующие файлы – такие как *трассировка стека* или *файл журнала* Tarantool.

Если это запрос новой функции или это затрагивает определенную группу пользователей, не забудьте это указать.

Обычно член команды Tarantool отвечает в течение одного-двух рабочих дней, чтобы подтвердить, что тикет взят в работу, задать уточняющие вопросы или предложить альтернативное решение описанной проблемы.

## 6.16 Руководство по разрешению проблем

### 6.16.1 Проблема: при выполнении INSERT/UPDATE-запросов возникает ошибка ER_MEMORY_ISSUE

**Возможные причины**

- Нехватка памяти (значения параметров `arena_used_ratio` и `quota_used_ratio` из *box.slab.info()* приближаются к 100%).

  Чтобы проверить значения данных параметров, выполните соответствующие команды:

```
$ # attaching to a Tarantool instance
$ tt connect <instance_name|URI>
```

```
-- запрашиваем значение arena_used_ratio
tarantool> box.slab.info().arena_used_ratio

-- запрашиваем значение quota_used_ratio
tarantool> box.slab.info().quota_used_ratio
```

**Решение**

У вас есть несколько вариантов действий:

- Зайти в *конфигурационный файл* Tarantool и увеличить значение параметра *box.cfg{memtx_memory}* (при наличии свободных ресурсов).

  В версиях Tarantool до 1.10 для изменения данного параметра требуется перезагрузить сервер. При обычной перезагрузке сервер будет недоступен на время старта Tarantool из .xlog-файлов. При перезагрузке в режиме горячего резервирования *hot standby* гарантирована практически 100%-ная доступность.

- Провести очистку базы данных.

- Проверьте, нет ли проблем с фрагментацией памяти:

```
-- запрашиваем значение quota_used_ratio
tarantool> box.slab.info().quota_used_ratio

-- запрашиваем значение items_used_ratio
tarantool> box.slab.info().items_used_ratio
```

  При высокой степени фрагментации памяти (значение параметра `quota_used_ratio` приближается к 100%, `items_used_ratio` около 50%) рекомендуется перезапустить Tarantool в режиме горячего резервирования *hot standby*.

### 6.16.2 Проблема: Tarantool создает большую нагрузку на CPU

**Возможные причины**

The *transaction processor thread* consumes over 60% CPU.

**Решение**

Attach to the Tarantool instance with *tt* utility, analyze the query statistics with *box.stat()* and spot the CPU consumption leader. The following commands can help:

```
$ # attaching to a Tarantool instance
$ tt connect <instance_name|URI>
```

```
-- запрашиваем RPS для вызовов хранимых процедур
tarantool> box.stat().CALL.rps
```

Критическое значение RPS – 75 000, в случае большого Lua-приложения (модульного приложения, содержащего более 200 строк кода) – 10 000 - 20 000.

```
-- запрашиваем RPS для запросов указанного типа
tarantool> box.stat().<query_type>.rps
```

Критическое значение RPS для запросов типа SELECT/INSERT/UPDATE/DELETE – 100 000.

Если основная нагрузка генерируется SELECT-запросами, следует добавить *slave-сервер* и часть запросов обрабатывать на нем.

Если же нагрузка по большей части приходится на INSERT/UPDATE/DELETE-запросы, рекомендуется провести *шардинг базы данных*.

### 6.16.3 Проблема: обработка запросов прекращается по таймауту

**Возможные причины**

---

**Примечание:** Все описанные ниже ситуации можно распознать по записям в журнале Tarantool, начинающимся со слов `'Too long...'`.

---

1. Быстрые и медленные запросы обрабатываются в одном подключении, что приводит к забиванию readahead-буфера медленными запросами.

   **Решение**

   У вас есть несколько вариантов действий:

   - Увеличить размер readahead-буфера (*box.cfg{readahead}*).

     This parameter can be changed on the fly, so you don't need to restart Tarantool. Attach to the Tarantool instance with *tt* utility and call `box.cfg{}` with a new `readahead` value:

     ```
     $ # attaching to a Tarantool instance
     $ tt connect <instance_name|URI>
     ```

     ```
     -- задаем новое значение readahead
     tarantool> box.cfg{readahead = 10 * 1024 * 1024}
     ```

     **Пример расчета:** при 1000 RPS, размере одного запроса в 1 Кбайт и максимальном времени обработки одного запроса в 10 секунд минимальный размер readahead-буфера должен равняться 10 Мбайт.

   - Обрабатывать быстрые и медленные запросы в отдельных подключениях (решается на уровне бизнес-логики).

2. Медленная работа дисков.

   **Решение**

   Проверить занятость дисков (с помощью утилиты iostat, iotop или strace посмотреть на параметр `iowait`) и попробовать разнести .xlog-файлы и снимки состояния базы данных по разным дискам (т.е. указать разные значения для параметров *wal_dir* и *memtx_dir*).

### 6.16.4 Проблема: параметры репликации lag и idle принимают отрицательные значения

Речь идет о параметрах `box.info.replication.(upstream.)lag` и `box.info.replication.(upstream.)idle` из сводной таблицы *box.info.replication*.

**Возможные причины**

Не синхронизированы часы на машинах или неправильно работает NTP-сервер.

**Решение**

Проверить настройки NTP-сервера.

Если проблем с NTP-сервером не обнаружено, то не следует ничего предпринимать, потому что при вычислении лага репликации используются показания системных часов на двух разных машинах, и в случае рассинхронизации может случиться так, что часы удаленного мастер-сервера всегда будут отставать от часов локального экземпляра Tarantool.

### 6.16.5 Проблема: значение параметра idle растет, но журнал не содержит связанных с этим сообщений

Речь идет о параметре `box.info.replication.(upstream.)idle` из сводной таблицы *box.info.replication*.

**Возможные причины**

Одному серверу были назначены различные IP-адреса или один и тот же сервер был указан в `box.cfg{}` дважды, что привело к установлению дублирующего подключения.

**Решение**

*Обновить Tarantool 1.6 до 1.7*, где эта ошибка была исправлена: в описанной ситуации репликация будет остановлена, а в журнал будет записана следующая ошибка: `'Incorrect value for option ''replication_source'': duplicate connection with the same replica UUID'`.

### 6.16.6 Проблема: общие параметры репликации не совпадают на репликах в рамках одного кластера

Речь идет о кластере, состоящем из одного мастера и нескольких реплик. В таком случае значения общих параметров из сводной таблицы *box.info.replication*, например `box.info.replication.lsn`, должны приходить с мастера и должны быть одинаковыми на всех репликах. Если такие параметры не совпадают, это свидетельствует о наличии проблем.

**Возможные причины**

Сбой репликации.

**Решение**

*Перезапустить репликацию*.

### 6.16.7 Проблема: репликация мастер-мастер остановлена

Речь идет о том, что параметр *box.info.replication(.upstream).status* имеет значение stopped.

**Возможные причины**

В репликационном кластере, состоящем из двух мастер-серверов, один из серверов попытался выполнить действие, уже выполненное другим сервером, – например, повторно вставить кортеж с таким же уникальным ключом (распознается по ошибке вида `'Duplicate key exists in unique index 'primary' in space <space_name>'`).

**Решение**

This issue can be fixed in two ways:

- Manually: *reseed* one master from another by removing write-ahead logs and snapshots.
- Programmatically: set up a *conflict resolution trigger*.

Then, restart replication as described in *Restarting replication*.

### 6.16.8 Проблема: Tarantool работает заметно медленнее, чем раньше

**Возможные причины**

Неэффективное использование памяти (память занята большим количеством неиспользуемых объектов).

**Решение**

Запустить сборщик мусора в Lua с помощью функции collectgarbage(count) и измерить время ее выполнения с помощью *clock.bench()* или *clock.proc()*.

Пример кода для подсчета потребляемой памяти:

```
$ # attaching to a Tarantool instance
$ tt connect <instance_name|URI>
```

```
-- загрузка модуля clock для работы со временем
tarantool> local clock = require 'clock'
-- запускаем таймер
tarantool> local b = clock.proc()
-- запускаем сборку мусора
tarantool> local c = collectgarbage('count')
-- останавливаем таймер по завершении сборки мусора
tarantool> return c, clock.proc() - b
```

Если возвращаемое `clock.proc()` значение больше 0.001, это может являться признаком неэффективного использования памяти (активного вмешательства не требуется, но рекомендуется оптимизация кода). Если значение превышает 0.01, необходимо провести подробный анализ кода и оптимизировать потребление памяти.

Если значение больше 0,01, код приложения однозначно необходимо проанализировать на предмет оптимизации использования памяти.

### 6.16.9 Проблема: Переключатель файберов запрещен в метаметоде `__gc`

#### Описание проблемы

Переключатель файберов запрещен в метаметоде `__gc`, начиная с этого изменения, во избежание неожиданной нехватки памяти в Lua. Однако может потребоваться функция передачи управления для финализации ресурсов, например, для закрытия сокета.

Ниже приведены примеры правильной реализации такой процедуры.

#### Решение

Для начала есть два простых примера, которые иллюстрируют логику решения:

- *Пример 1*
- *Пример 2*.

Далее идет *Пример 3*, который проиллюстрирует использование модуля `sched.lua`, — рекомендуемый метод.

Все пояснения приведены в комментариях в листинге кода. `-- >` обозначает вывод в консоль.

**Пример 1**

Реализация подходящего финализатора для определенного типа FFI (custom_t).

```
local ffi = require('ffi')
local fiber = require('fiber')

ffi.cdef('struct custom { int a; };')

local function __custom_gc(self)
  print(("Entered custom GC finalizer for %s... (before yield)"):format(self.a))
  fiber.yield()
  print(("Leaving custom GC finalizer for %s... (after yield)"):format(self.a))
end

local custom_t = ffi.metatype('struct custom', {
  __gc = function(self)
    -- XXX: Do not invoke yielding functions in __gc metamethod.
    -- Create a new fiber to run after the execution leaves
    -- this routine.
    fiber.new(__custom_gc, self)
    print(("Finalization is scheduled for %s..."):format(self.a))
  end
})

-- Create a cdata object of <custom_t> type.
local c = custom_t(42)

-- Remove a single reference to that object to make it subject
-- for GC.
c = nil

-- Run full GC cycle to purge the unreferenced object.
collectgarbage('collect')
-- > Finalization is scheduled for 42...

-- XXX: There is no finalization made until the running fiber
-- yields its execution. Let's do it now.
fiber.yield()
-- > Entered custom GC finalizer for 42... (before yield)
-- > Leaving custom GC finalizer for 42... (after yield)
```

**Пример 2**

Implementing a valid finalizer for a particular user type (struct custom).

custom.c

```
#include <lauxlib.h>
#include <lua.h>
#include <module.h>
#include <stdio.h>

struct custom {
  int a;
};
```

(continues on next page)

(продолжение с предыдущей страницы)

```
const char *CUSTOM_MTNAME = "CUSTOM_MTNAME";

/*
 * XXX: Do not invoke yielding functions in __gc metamethod.
 * Create a new fiber to be run after the execution leaves
 * this routine. Unfortunately we can't pass the parameters to the
 * routine to be executed by the created fiber via <fiber_new_ex>.
 * So there is a workaround to load the Lua code below to create
 * __gc metamethod passing the object for finalization via Lua
 * stack to the spawned fiber.
 */
const char *gc_wrapper_constructor = " local fiber = require('fiber')         "
             " print('constructor is initialized')    "
             " return function(__custom_gc)           "
             "   print('constructor is called')       "
             "   return function(self)                "
             "     print('__gc is called')            "
             "     fiber.new(__custom_gc, self)       "
             "     print('Finalization is scheduled') "
             "   end                                  "
             " end                                    "
        ;

int custom_gc(lua_State *L) {
  struct custom *self = luaL_checkudata(L, 1, CUSTOM_MTNAME);
  printf("Entered custom_gc for %d... (before yield)\n", self->a);
  fiber_sleep(0);
  printf("Leaving custom_gc for %d... (after yield)\n", self->a);
  return 0;
}

int custom_new(lua_State *L) {
  struct custom *self = lua_newuserdata(L, sizeof(struct custom));
  luaL_getmetatable(L, CUSTOM_MTNAME);
  lua_setmetatable(L, -2);
  self->a = lua_tonumber(L, 1);
  return 1;
}

static const struct luaL_Reg libcustom_methods [] = {
  { "new", custom_new },
  { NULL, NULL }
};

int luaopen_custom(lua_State *L) {
  int rc;

  /* Create metatable for struct custom type */
  luaL_newmetatable(L, CUSTOM_MTNAME);
  /*
   * Run the constructor initializer for GC finalizer:
   * - load fiber module as an upvalue for GC finalizer
```

(continues on next page)

(продолжение с предыдущей страницы)

```
   *   constructor
   * - return GC finalizer constructor on the top of the
   *   Lua stack
   */
  rc = luaL_dostring(L, gc_wrapper_constructor);
  /*
   * Check whether constructor is initialized (i.e. neither
   * syntax nor runtime error is raised).
   */
  if (rc != LUA_OK)
    luaL_error(L, "test module loading failed: constructor init");
  /*
   * Create GC object for <custom_gc> function to be called
   * in scope of the GC finalizer and push it on top of the
   * constructor returned before.
   */
  lua_pushcfunction(L, custom_gc);
  /*
   * Run the constructor with <custom_gc> GCfunc object as
   * a single argument. As a result GC finalizer is returned
   * on the top of the Lua stack.
   */
  rc = lua_pcall(L, 1, 1, 0);
  /*
   * Check whether GC finalizer is created (i.e. neither
   * syntax nor runtime error is raised).
   */
  if (rc != LUA_OK)
    luaL_error(L, "test module loading failed: __gc init");
  /*
   * Assign the returned function as a __gc metamethod to
   * custom type metatable.
   */
  lua_setfield(L, -2, "__gc");

  /*
   * Initialize Lua table for custom module and fill it
   * with the custom methods.
   */
  lua_newtable(L);
  luaL_register(L, NULL, libcustom_methods);
  return 1;
}
```

custom_c.lua

```
-- Load custom Lua C extension.
local custom = require('custom')
-- > constructor is initialized
-- > constructor is called

-- Create a userdata object of <struct custom> type.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
local c = custom.new(9)

-- Remove a single reference to that object to make it subject
-- for GC.
c = nil

-- Run full GC cycle to purge the unreferenced object.
collectgarbage('collect')
-- > __gc is called
-- > Finalization is scheduled

-- XXX: There is no finalization made until the running fiber
-- yields its execution. Let's do it now.
require('fiber').yield()
-- > Entered custom_gc for 9... (before yield)

-- XXX: Finalizer yields the execution, so now we are here.
print('We are here')
-- > We are here

-- XXX: This fiber finishes its execution, so yield to the
-- remaining fiber to finish the postponed finalization.
-- > Leaving custom_gc for 9... (after yield)
```

**Example 3**

It is important to note that the finalizer implementations in the examples above increase pressure on the platform performance by creating a new fiber on each `__gc` call. To prevent such an excessive fibers spawning, it's better to start a single «scheduler» fiber and provide the interface to postpone the required asynchronous action.

For this purpose, the module called `sched.lua` is implemented (see the listing below). It is a part of Tarantool and should be made required in your custom code. The usage example is given in the `init.lua` file below.

`sched.lua`

```
local fiber = require('fiber')

local worker_next_task = nil
local worker_last_task
local worker_fiber
local worker_cv = fiber.cond()

-- XXX: the module is not ready for reloading, so worker_fiber is
-- respawned when sched.lua is purged from package.loaded.

--
-- Worker is a singleton fiber for not urgent delayed execution of
-- functions. Main purpose - schedule execution of a function,
-- which is going to yield, from a context, where a yield is not
-- allowed. Such as an FFI object's GC callback.
--
local function worker_f()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  while true do
    local task
    while true do
      task = worker_next_task
      if task then break end
      -- XXX: Make the fiber wait until the task is added.
      worker_cv:wait()
    end
    worker_next_task = task.next
    task.f(task.arg)
    fiber.yield()
  end
end

local function worker_safe_f()
  pcall(worker_f)
  -- The function <worker_f> never returns. If the execution is
  -- here, this fiber is probably canceled and now is not able to
  -- sleep. Create a new one.
  worker_fiber = fiber.new(worker_safe_f)
end

worker_fiber = fiber.new(worker_safe_f)

local function worker_schedule_task(f, arg)
  local task = { f = f, arg = arg }
  if not worker_next_task then
    worker_next_task = task
  else
    worker_last_task.next = task
  end
  worker_last_task = task
  worker_cv:signal()
end

return {
  postpone = worker_schedule_task
}
```

init.lua

```
local ffi = require('ffi')
local fiber = require('fiber')
local sched = require('sched')

local function __custom_gc(self)
  print(("Entered custom GC finalizer for %s... (before yield)"):format(self.a))
  fiber.yield()
  print(("Leaving custom GC finalizer for %s... (after yield)"):format(self.a))
end

ffi.cdef('struct custom { int a; };')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
local custom_t = ffi.metatype('struct custom', {
  __gc = function(self)
    -- XXX: Do not invoke yielding functions in __gc metamethod.
    -- Schedule __custom_gc call via sched.postpone to be run
    -- after the execution leaves this routine.
    sched.postpone(__custom_gc, self)
    print(("Finalization is scheduled for %s..."):format(self.a))
  end
})

-- Create several <custom_t> objects to be finalized later.
local t = { }
for i = 1, 10 do t[i] = custom_t(i) end

-- Run full GC cycle to collect the existing garbage. Nothing is
-- going to be printed, since the table <t> is still "alive".
collectgarbage('collect')

-- Remove the reference to the table and, ergo, all references to
-- the objects.
t = nil

-- Run full GC cycle to collect the table and objects inside it.
-- As a result all <custom_t> objects are scheduled for further
-- finalization, but the finalizer itself (i.e. __custom_gc
-- functions) is not called.
collectgarbage('collect')
-- > Finalization is scheduled for 10...
-- > Finalization is scheduled for 9...
-- > ...
-- > Finalization is scheduled for 2...
-- > Finalization is scheduled for 1...

-- XXX: There is no finalization made until the running fiber
-- yields its execution. Let's do it now.
fiber.yield()
-- > Entered custom GC finalizer for 10... (before yield)

-- XXX: Oops, we are here now, since the scheduler fiber yielded
-- the execution to this one. Check this out.
print("We're here now. Let's continue the scheduled finalization.")
-- > We're here now. Let's continue the finalization

-- OK, wait a second to allow the scheduler to cleanup the
-- remaining garbage.
fiber.sleep(1)
-- > Leaving custom GC finalizer for 10... (after yield)
-- > Entered custom GC finalizer for 9... (before yield)
-- > Leaving custom GC finalizer for 9... (after yield)
-- > ...
-- > Entered custom GC finalizer for 1... (before yield)
-- > Leaving custom GC finalizer for 1... (after yield)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print("Did we finish? I guess so.")
-- > Did we finish? I guess so.

-- Stop the instance.
os.exit(0)
```

## 6.17 Monitoring

Monitoring is the process of measuring and tracking Tarantool performance based on metrics. The metrics are typically monitored in real time, which allows you to identify or predict issues.

This chapter includes the following sections:

### 6.17.1 Getting started with monitoring

If you use Tarantool version below 2.11.1, it is necessary to install the latest version of `metrics` first. For details, see *Installing the metrics module*.

#### Using the metrics module

---

**Примечание:** The module is also used in applications based on the Cartridge framework. For details, see the *Getting started with Cartridge application* section.

---

1. First, set the instance name and start to collect the standard set of metrics:

   ```
   metrics.cfg{labels = {alias = 'my-instance'}}
   ```

   When using a metrics module version below **0.17.0**, use the following snippet instead of `metrics.cfg(...)`:

   ```
   metrics.set_global_labels({alias = 'my-instance'})
   metrics.enable_default_metrics()
   ```

2. Then, add a handler to expose metric values.

   For JSON format:

   ```
   local json_exporter = require('metrics.plugins.json')
   local function http_metrics_handler(request)
       return request:render({ text = json_exporter.export() })
   end
   ```

   For Prometheus format:

   ```
   local prometheus_exporter = require('metrics.plugins.prometheus').collect_http
   ```

   To learn how to extend metrics with custom data, check the *API reference*.

3. Start the HTTP server and expose metrics:

```
local http_server = require('http.server')
local server = http_server.new('0.0.0.0', 8081)
server:route({path = '/metrics'}, http_metrics_handler)
server:start()
```

The metric values are now available via the `http://localhost:8081/metrics` URL:


```
[
  {
    "label_pairs": {
      "alias": "my-instance"
    },
    "timestamp": 1679663602823779,
    "metric_name": "tnt_vinyl_disk_index_size",
    "value": 0
  },
  {
    "label_pairs": {
      "alias": "my-instance"
    },
    "timestamp": 1679663602823779,
    "metric_name": "tnt_info_memory_data",
    "value": 39272
  },
  {
    "label_pairs": {
      "alias": "my-instance"
    },
    "timestamp": 1679663602823779,
    "metric_name": "tnt_election_vote",
    "value": 0
  }
]
```


The data can be visualized in Grafana dashboard.

The full source example is listed below:

```
-- Import modules
local metrics = require('metrics')
local http_server = require('http.server')
local json_exporter = require('metrics.plugins.json')

-- Define helper functions
local function http_metrics_handler(request)
    return request:render({ text = json_exporter.export() })
end

-- Start the database
box.cfg{
    listen = 3301,
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Configure the metrics module
metrics.cfg{labels = {alias = 'my-tnt-app'}}

-- Run the web server
local server = http_server.new('0.0.0.0', 8081)
server:route({path = '/metrics'}, http_metrics_handler)
server:start()
```

### Collecting HTTP metrics

To enable the collection of HTTP metrics, wrap a handler with a `metrics.http_middleware.v1` function:

```
local metrics = require('metrics')
local httpd = require('http.server').new(ip, port)

-- Create a summary collector for latency
metrics.http_middleware.configure_default_collector('summary')

-- Set a route handler for latency summary collection
httpd:route({ path = '/path-1', method = 'POST' }, metrics.http_middleware.v1(handler_1))
httpd:route({ path = '/path-2', method = 'GET' }, metrics.http_middleware.v1(handler_2))

-- Start HTTP routing
httpd:start()
```

---

**Примечание:** By default, the `http_middleware` uses the *histogram* collector for backward compatibility reasons. To collect HTTP metrics, use the *summary* type instead.

---

You can collect all HTTP metrics with a single collector. If you use the default *Grafana dashboard*, don't change the default collector name. Otherwise, your metrics won't appear on the charts.

### Creating custom metric

You can create your own metric in two ways, depending on when you need to take measurements:

- at any arbitrary moment of time
- when the data collected by metrics is requested

To create custom metrics at any arbitrary moment of time, do the following:

1. Create the collector:

   ```
   local response_counter = metrics.counter('response_counter', 'Response counter')
   ```

2. Take a measurement at the appropriate place, for example, in an API request handler:

   ```
   local function check_handler(request)
       local label_pairs = {
           path = request.path,
           method = request.method,
   ```

(continues on next page)

(продолжение с предыдущей страницы)

```
    }
    response_counter:inc(1, label_pairs)
    -- ...
end
```

To create custom metrics when the data collected by metrics is requested, do the following:

1. Create the collector:

```
local other_custom_metric = metrics.gauge('other_custom_metric', 'Other custom␣
↪metric')
```

2. Take a measurement at the time of requesting the data collected by metrics:

```
metrics.register_callback(function()
    -- ...
    local label_pairs = {
        category = category,
    }
    other_custom_metric:set(current_value, label_pairs)
end)
```

The full example is listed below:

```
-- Import modules
local metrics = require('metrics')
local http_server = require('http.server')
local json_exporter = require('metrics.plugins.json')

local response_counter = metrics.counter('response_counter', 'Response counter')

-- Define helper functions
local function http_metrics_handler(request)
    return request:render({ text = json_exporter.export() })
end

local function check_handler(request)
    local label_pairs = {
        path = request.path,
        method = request.method,
    }
    response_counter:inc(1, label_pairs)
    return request:render({ text = 'ok' })
end

-- Start the database
box.cfg{
    listen = 3301,
}

-- Configure the metrics module
metrics.set_global_labels{alias = 'my-tnt-app'}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Run the web server
local server = http_server.new('0.0.0.0', 8081)
server:route({path = '/metrics'}, http_metrics_handler)
server:route({path = '/check'}, check_handler)
server:start()
```

The result looks in the following way:

```
[
  {
    "label_pairs": {
      "path": "/check",
      "method": "GET",
      "alias": "my-tnt-app"
    },
    "timestamp": 1688385933874080,
    "metric_name": "response_counter",
    "value": 1
  }
]
```

### Possible limitations

The module allows to add your own metrics, but there are some subtleties when working with specific tools.

When adding your custom metric, it's important to ensure that the number of label value combinations is kept to a minimum. Otherwise, combinatorial explosion may happen in the timeseries database with metrics values stored. Examples of data labels:

- Labels in Prometheus
- Tags in InfluxDB

For example, if your company uses InfluxDB for metric collection, you could potentially disrupt the entire monitoring setup, both for your application and for all other systems within the company. As a result, monitoring data is likely to be lost.

Example:

```
local some_metric = metrics.counter('some', 'Some metric')

-- THIS IS POSSIBLE
local function on_value_update(instance_alias)
   some_metric:inc(1, { alias = instance_alias })
end

-- THIS IS NOT ALLOWED
local function on_value_update(customer_id)
   some_metric:inc(1, { customer_id = customer_id })
end
```

In the example, there are two versions of the function `on_value_update`. The top version labels the data with the cluster instance's alias. Since there's a relatively small number of nodes, using them as labels is

feasible. In the second case, an identifier of a record is used. If there are many records, it's recommended to avoid such situations.

The same principle applies to URLs. Using the entire URL with parameters is not recommended. Use a URL template or the name of the command instead.

In essence, when designing custom metrics and selecting labels or tags, it's crucial to opt for a minimal set of values that can uniquely identify the data without introducing unnecessary complexity or potential conflicts with existing metrics and systems.

### 6.17.2 Installing the metrics module

**Примечание:** Since Tarantool version 2.11.1, the installation is not required.

#### Installing metrics using the .rockspec file

Usually, all dependencies are included in the `*.rockspec` file of the application. All dependencies are installed from this file. To do this:

1. Add the `metrics` module to the dependencies in the `.rockspec` file:

```
dependencies = {
    ...
    'metrics == 1.0.0',
    ...
}
```

2. Install the missing dependencies:

```
tt rocks make
# OR #
cartridge build
```

#### Installing the metrics module only

To install only the `metrics` module, execute the following commands:

1. Set current folder:

```
$ cd ${PROJECT_ROOT}
```

2. Install the missing dependencies:

```
$ tt rocks install metrics <version>
```

   where `version` – the necessary version number. If omitted, then the version from the `master` branch is installed.

### 6.17.3 Getting started with Cartridge application

The `metrics` module is tightly integrated with the Cartridge framework. To enable this integration, a permanent role called `metrics` has been introduced. To enable this role, follow the steps below.

#### Module setup

First, install the latest version of `metrics`. For details, *check the installation guide*.

Also, you need to install the separate `cartridge-metrics-role` rock. To do this:

1. Add the `cartridge-metrics-role` package to the dependencies in the `.rockspec` file:

```
dependencies = {
    ...
    'cartridge-metrics-role',
    ...
}
```

2. Next, install the missing dependencies:

```
tt rocks make
# OR #
cartridge build
```

After the `cartridge-metrics-role` installation, enable this package in the list of roles in `cartridge.cfg`:

```
local ok, err = cartridge.cfg({
    roles = {
        ...
        'cartridge.roles.metrics',
        ...
    },
})
```

Then, configure the `metrics` module in either of two ways:

- add the `metrics` configuration section to your cluster configuration;
- specify the configuration in the separate `metrics.yml` file.

In the configuration, specify the response format and the addresses at which the commands are available:

```
metrics:
  export:
    - path: '/path_for_json_metrics'
      format: 'json'
    - path: '/path_for_prometheus_metrics'
      format: 'prometheus'
    - path: '/health'
      format: 'health'
```

Learn more about Cartridge configuration.

**Примечание:** Instead of configuring the cluster configuration, you can also use the set_export command.

Now the commands“ data is accessible at the following addresses:

```
http://url:port/path_for_json_metrics
http://url:port/path_for_prometheus_metrics
http://url:port/health
```

where `url:port` – the address and Cartridge HTTP port of a specific instance of the application.

You can visualize the data in Grafana dashboard.

After the role has been initialized, the default metrics are enabled and the global `alias` label is set.

---

**Примечание:** Since **0.6.1**, the `alias` label value is set by the `alias` or `instance_name` instance configuration option.

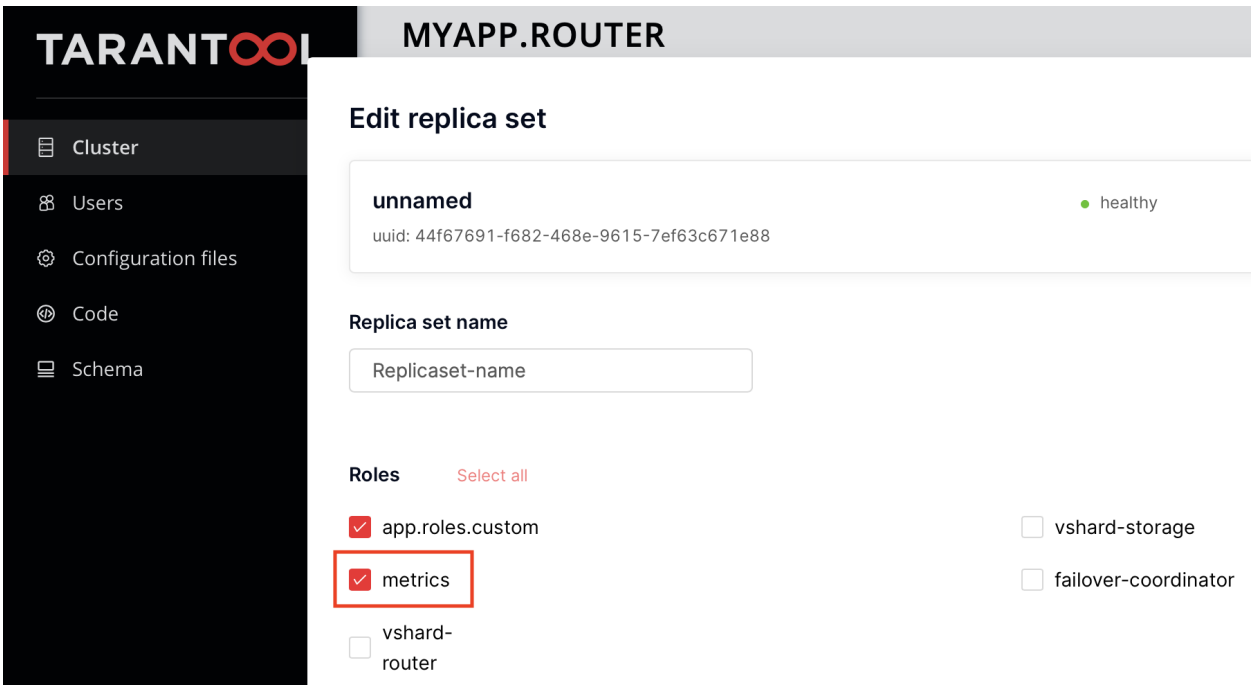
---

You can use the functionality of any metrics package. To do this, get the package as a Cartridge service and call it with the `require()` like a regular package:

```
local cartridge = require('cartridge')
local metrics = cartridge.service_get('metrics')
```

### Additional steps for older versions of the metrics module

Since version **0.6.0**, the `metrics` role is permanent and enabled on instances by default. If you use an old version of `metrics`, you need to enable the role in the interface first:



### Adding metrics to HTTP API commands of the application

You can connect the standard `http_server_request_latency` metric to your application's HTTP API commands. This metric records the number of invocations and the total execution time (latency) of each individual command. To connect this, wrap each API handler with the `metrics.http_middleware.v1(...)` function.

Example:

```
local cartridge = require('cartridge')
local server = cartridge.service_get('httpd') -- get the HTTP server module
local metrics = cartridge.service_get('metrics') -- get the module of metrics

local function http_app_api_handler(request) -- add test command
    return request:render({ text = 'Hello world!!!' })
end

local server = http_server.new('0.0.0.0', 8081)
server:route({path = '/hello'}, metrics.http_middleware.v1(http_app_api_handler))
server:start()
```

When calling the `cartridge.service_get('metrics')` command as an application (usually in a router), add a dependency of this role on the role of `metrics`:

```
return {
    ...
    dependencies = {
        ...
        'cartridge.roles.metrics',
    }
}
```

Now after the HTTP API calls `hello` at `http://url:port/path_for_json_metrics`, new data on these calls is available:

```
{
    "label_pairs": {
        "path": "/hello",
        "method": "ANY",
        "status": 200,
        "alias": "my-tnt-app"
    },
    "timestamp": 1679668258972227,
    "metric_name": "http_server_request_latency_count",
    "value": 9
},
{
    "label_pairs": {
        "path": "/hello",
        "method": "ANY",
        "status": 200,
        "alias": "my-tnt-app"
    },
    "timestamp": 1679668258972227,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "metric_name": "http_server_request_latency_sum",
    "value": 0.00008015199273359
},
```

The default type for this metric is `histogram`. However, it's *recommended* to use the `summary` type instead.

### Additional settings

- Since Tarantool Cartridge `2.4.0`, you can set a zone for each instance in the cluster. When a zone is set, all the metrics on the instance receive the `zone` label.
- You can set custom global labels with the following configuration:

  ```
  metrics:
    export:
      - path: '/metrics'
        format: 'json'
    global-labels:
      my-custom-label: label-value
  ```

  Another option is to invoke the `set_default_labels` function in `init.lua`:

  ```
  local metrics = require('cartridge.roles.metrics')
  metrics.set_default_labels({ ['my-custom-label'] = 'label-value' })
  ```

- You can use the configuration below to choose the default metrics to be exported. If you add the `include` section, only the metrics from this section will be exported:

  ```
  metrics:
    export:
      - path: '/metrics'
        format: 'json'
    # export only vinyl, luajit and memory metrics:
    include:
      - vinyl
      - luajit
      - memory
  ```

  If you add the `exclude` section, the metrics from this section will be removed from the default metrics list:

  ```
  metrics:
    export:
      - path: '/metrics'
        format: 'json'
    # export all metrics except vinyl, luajit and memory:
    exclude:
      - vinyl
      - luajit
      - memory
  ```

  For the full list of default metrics, check the *API reference*.

#### Creating a custom health check format

By default, the response of the health command contains a status code of

- 200 – if everything is okay,
- 500 – if the instance is unhealthy.

You can set your own response format in the following way:

```
local health = require('cartridge.health')
local metrics = cartridge.service_get('metrics')

metrics.set_health_handler(function(req)
    local resp = req:render{
        json = {
            my_healthcheck_format = health.is_healthy()
        }
    }
    resp.status = 200
    return resp
end)
```

### 6.17.4 Metrics reference

This page provides a detailed description of metrics from the `metrics` module.

#### General metrics

General instance information:

| | |
|---|---|
| `tnt_cfg_current_time` | Instance system time in the Unix timestamp format |
| `tnt_info_uptime` | Time in seconds since the instance has started |
| `tnt_read_only` | Indicates if the instance is in read-only mode (`1` if true, `0` if false) |

#### Memory general

The following metrics provide a picture of memory usage by the Tarantool process.

| | |
|---|---|
| `tnt_info_memory_cache` | Number of bytes in the cache used to store tuples with the vinyl storage engine. |
| `tnt_info_memory_data` | Number of bytes used to store user data (tuples) with the memtx engine and with level 0 of the vinyl engine, without regard for memory fragmentation. |
| `tnt_info_memory_index` | Number of bytes used for indexing user data. Includes memtx and vinyl memory tree extents, the vinyl page index, and the vinyl bloom filters. |
| `tnt_info_memory_lua` | Number of bytes used for the Lua runtime. The Lua memory is limited to 2 GB per instance. Monitoring this metric can prevent memory overflow. |
| `tnt_info_memory_net` | Number of bytes used for network input/output buffers. |
| `tnt_info_memory_tx` | Number of bytes in use by active transactions. For the vinyl storage engine, this is the total size of all allocated objects (struct `txv`, struct `vy_tx`, struct `vy_read_interval`) and tuples pinned for those objects. |

### Memory allocation

Provides a memory usage report for the slab allocator. The slab allocator is the main allocator used to store tuples. The following metrics help monitor the total memory usage and memory fragmentation. To learn more about use cases, refer to the *box.slab submodule documentation*.

Available memory, bytes:

| | |
|---|---|
| `tnt_slab_quota_size` | Amount of memory available to store tuples and indexes. Is equal to `memtx_memory`. |
| `tnt_slab_arena_size` | Total memory available to store both tuples and indexes. Includes allocated but currently free slabs. |
| `tnt_slab_items_size` | Total amount of memory available to store only tuples and not indexes. Includes allocated but currently free slabs. |

Memory usage, bytes:

| | |
|---|---|
| `tnt_slab_quota_used` | The amount of memory that is already reserved by the slab allocator. |
| `tnt_slab_arena_used` | The effective memory used to store both tuples and indexes. Disregards allocated but currently free slabs. |
| `tnt_slab_items_used` | The effective memory used to store only tuples and not indexes. Disregards allocated but currently free slabs. |

Memory utilization, %:

| | |
|---|---|
| `tnt_slab_quota_used_ratio` | `tnt_slab_quota_used / tnt_slab_quota_size` |
| `tnt_slab_arena_used_ratio` | `tnt_slab_arena_used / tnt_slab_arena_size` |
| `tnt_slab_items_used_ratio` | `tnt_slab_items_used / tnt_slab_items_size` |

### Spaces

The following metrics provide specific information about each individual space in a Tarantool instance.

| | |
|---|---|
| `tnt_space_len` | Number of records in the space. This metric always has 2 labels: `{name="test", engine="memtx"}`, where `name` is the name of the space and `engine` is the engine of the space. |
| `tnt_space_bsize` | Total number of bytes in all tuples. This metric always has 2 labels: `{name="test", engine="memtx"}`, where `name` is the name of the space and `engine` is the engine of the space. |
| `tnt_space_index_bsize` | Total number of bytes taken by the index. This metric always has 2 labels: `{name="test", index_name="pk"}`, where `name` is the name of the space and `index_name` is the name of the index. |
| `tnt_space_total_bsize` | Total size of tuples and all indexes in the space. This metric always has 2 labels: `{name="test", engine="memtx"}`, where `name` is the name of the space and `engine` is the engine of the space. |
| `tnt_vinyl_tuples` | Total tuple count for vinyl. This metric always has 2 labels: `{name="test", engine="vinyl"}`, where `name` is the name of the space and `engine` is the engine of the space. For vinyl this metric is disabled by default and can be enabled only with global variable setup: `rawset(_G, 'include_vinyl_count', true)`. |

### Network

Network activity stats. These metrics can be used to monitor network load, usage peaks, and traffic drops.

Sent bytes:

| | |
|---|---|
| `tnt_net_sent_total` | Bytes sent from the instance over the network since the instance's start time |

Received bytes:

| | |
|---|---|
| `tnt_net_received_total` | Bytes received by the instance since start time |

Connections:

| | |
|---|---|
| `tnt_net_connections_total` | Number of incoming network connections since the instance's start time |
| `tnt_net_connections_current` | Number of active network connections |

Requests:

| | |
|---|---|
| `tnt_net_requests_total` | Number of network requests the instance has handled since its start time |
| `tnt_net_requests_current` | Number of pending network requests |

Requests in progress:

| | |
|---|---|
| `tnt_net_requests_in_progress_total` | Total number of requests processed by tx thread |
| `tnt_net_requests_in_progress_current` | Count of requests currently being processed in the tx thread |

Requests placed in queues of streams:

| | |
|---|---|
| `tnt_net_requests_in_stream_total` | Total number of requests, which was placed in queues of streams for all time |
| `tnt_net_requests_in_stream_current` | Count of requests currently waiting in queues of streams |

Since Tarantool 2.10 in each network metric has the label `thread`, showing per-thread network statistics.

### Fibers

Provides the statistics for *fibers*. If your application creates a lot of fibers, you can use the metrics below to monitor fiber count and memory usage.

| | |
|---|---|
| `tnt_fiber_amount` | Number of fibers |
| `tnt_fiber_csw` | Overall number of fiber context switches |
| `tnt_fiber_memalloc` | Amount of memory reserved for fibers |
| `tnt_fiber_memused` | Amount of memory used by fibers |

### Operations

You can collect iproto requests an instance has processed and aggregate them by request type. This may help you find out what operations your clients perform most often.

| | |
|---|---|
| `tnt_stats_op_total` | Total number of calls since server start |

To distinguish between request types, this metric has the `operation` label. For example, it can look as follows: `{operation="select"}`. For the possible request types, check the table below.

| | |
|---|---|
| `auth` | Authentication requests |
| `call` | Requests to execute stored procedures |
| `delete` | Delete calls |
| `error` | Requests resulted in an error |
| `eval` | Calls to evaluate Lua code |
| `execute` | Execute SQL calls |
| `insert` | Insert calls |
| `prepare` | SQL prepare calls |
| `replace` | Replace calls |
| `select` | Select calls |
| `update` | Update calls |
| `upsert` | Upsert calls |

### Replication

Provides the current replication status. Learn more about *replication in Tarantool*.

| | |
|---|---|
| `tnt_info_lsn` | LSN of the instance. |
| `tnt_info_vclock` | LSN number in vclock. This metric always has the label `{id="id"}`, where `id` is the instance's number in the replica set. |
| `tnt_replication_lsn` | LSN of the tarantool instance. This metric always has labels `{id="id", type="type"}`, where `id` is the instance's number in the replica set, `type` is `master` or `replica`. |
| `tnt_replication_lag` | Replication lag value in seconds. This metric always has labels `{id="id", stream="stream"}`, where `id` is the instance's number in the replica set, `stream` is `downstream` or `upstream`. |
| `tnt_replication_status` | This metrics equals 1 when replication status is «follow» and 0 otherwise. This metric always has labels `{id="id", stream="stream"}`, where `id` is the instance's number in the replica set, `stream` is `downstream` or `upstream`. |

### Runtime

| | |
|---|---|
| `tnt_runtime_lua` | Lua garbage collector size in bytes |
| `tnt_runtime_used` | Number of bytes used for the Lua runtime |
| `tnt_runtime_tuple` | Number of bytes used for the tuples (except tuples owned by memtx and vinyl) |

### Cartridge

| | |
|---|---|
| `tnt_cartridge_issues` | Number of instance issues. This metric always has the label `{level="critical"}`, where `level` is the level of the issue:<br>• `critical` is associated with critical instance problems, such as the case when there is more than 90% memory used.<br>• `warning` is associated with other cluster problems, such as replication issues on the instance. |
| `tnt_cartridge_cluster_issues` | Sum of instance issues number over cluster. |
| `tnt_clock_delta` | Clock drift across the cluster. This metric always has the label `{delta="..."}`, which has the following possible values:<br>• `max` – difference with the fastest clock (always positive),<br>• `min` – difference with the slowest clock (always negative). |
| `tnt_cartridge_failover_trigger_total` | Count of failover triggers in cluster. |

### LuaJIT metrics

LuaJIT metrics provide an insight into the work of the Lua garbage collector. These metrics are available in Tarantool 2.6 and later.

General JIT metrics:

| | |
|---|---|
| `lj_jit_snap_restore_total` | Overall number of snap restores |
| `lj_jit_trace_num` | Number of JIT traces |
| `lj_jit_trace_abort_total` | Overall number of abort traces |
| `lj_jit_mcode_size` | Total size of allocated machine code areas |

JIT strings:

| | |
|---|---|
| `lj_strhash_hit_total` | Number of strings being interned |
| `lj_strhash_miss_total` | Total number of string allocations |

GC steps:

| | |
|---|---|
| `lj_gc_steps_atomic_total` | Count of incremental GC steps (atomic state) |
| `lj_gc_steps_sweepstring_total` | Count of incremental GC steps (sweepstring state) |
| `lj_gc_steps_finalize_total` | Count of incremental GC steps (finalize state) |
| `lj_gc_steps_sweep_total` | Count of incremental GC steps (sweep state) |
| `lj_gc_steps_propagate_total` | Count of incremental GC steps (propagate state) |
| `lj_gc_steps_pause_total` | Count of incremental GC steps (pause state) |

Allocations:

| | |
|---|---|
| `lj_gc_strnum` | Number of allocated `string` objects |
| `lj_gc_tabnum` | Number of allocated `table` objects |
| `lj_gc_cdatanum` | Number of allocated `cdata` objects |
| `lj_gc_udatanum` | Number of allocated `udata` objects |
| `lj_gc_freed_total` | Total amount of freed memory |
| `lj_gc_memory` | Current allocated Lua memory |
| `lj_gc_allocated_total` | Total amount of allocated memory |

### CPU metrics

The following metrics provide CPU usage statistics. They are only available on Linux.

| | |
|---|---|
| `tnt_cpu_number` | Total number of processors configured by the operating system |
| `tnt_cpu_time` | Host CPU time |
| `tnt_cpu_thread` | Tarantool thread CPU time. This metric always has the labels `{kind="user", thread_name="tarantool", thread_pid="pid", file_name="init.lua"}`, where:<br>• `kind` can be either `user` or `system`<br>• `thread_name` is `tarantool`, `wal`, `iproto`, or `coio`<br>• `file_name` is the entrypoint file name, for example, `init.lua`. |

There are also two cross-platform metrics, which can be obtained with a `getrusage()` call.

| | |
|---|---|
| `tnt_cpu_user_time` | Tarantool CPU user time |
| `tnt_cpu_system_time` | Tarantool CPU system time |

### Vinyl

Vinyl metrics provide *vinyl engine* statistics.

### Disk

The disk metrics are used to monitor overall data size on disk.

| | |
|---|---|
| `tnt_vinyl_disk_data_size` | Amount of data in bytes stored in the `.run` files located in *vinyl_dir* |
| `tnt_vinyl_disk_index_size` | Amount of data in bytes stored in the `.index` files located in *vinyl_dir* |

### Regulator

The vinyl regulator decides when to commence disk IO actions. It groups activities in batches so that they are more consistent and efficient.

| | |
|---|---|
| tnt_vinyl_regulator_dump_bandwidth | Estimated average dumping rate, bytes per second. The rate value is initially 10485760 (10 megabytes per second). It is recalculated depending on the the actual rate. Only significant dumps that are larger than 1 MB are used for estimating. |
| tnt_vinyl_regulator_write_rate | Actual average rate of performing write operations, bytes per second. The rate is calculated as a 5-second moving average. If the metric value is gradually going down, this can indicate disk issues. |
| tnt_vinyl_regulator_rate_limit | Write rate limit, bytes per second. The regulator imposes the limit on transactions based on the observed dump/compaction performance. If the metric value is down to approximately 10^5, this indicates issues with the disk or the *scheduler*. |
| tnt_vinyl_regulator_dump_watermark | Maximum amount of memory in bytes used for in-memory storing of a vinyl LSM tree. When this maximum is accessed, a dump must occur. For details, see *Наполнение LSM-дерева*. The value is slightly smaller than the amount of memory allocated for vinyl trees, reflected in the *vinyl_memory* parameter. |
| tnt_vinyl_regulator_blocked_writers | The number of fibers that are blocked waiting for Vinyl level0 memory quota. |

### Transactional activity

| | |
|---|---|
| tnt_vinyl_tx_commit | Counter of commits (successful transaction ends) Includes implicit commits: for example, any insert operation causes a commit unless it is within a *box.begin()–box.commit()* block. |
| tnt_vinyl_tx_rollback | Counter of rollbacks (unsuccessful transaction ends). This is not merely a count of explicit *box.rollback()* requests – it includes requests that ended with errors. |
| tnt_vinyl_tx_conflict | Counter of conflicts that caused transactions to roll back. The ratio tnt_vinyl_tx_conflict / tnt_vinyl_tx_commit above 5% indicates that vinyl is not healthy. At that moment, you'll probably see a lot of other problems with vinyl. |
| tnt_vinyl_tx_read_views | Current number of read views – that is, transactions that entered the read-only state to avoid conflict temporarily. Usually the value is 0. If it stays non-zero for a long time, it is indicative of a memory leak. |

### Memory

The following metrics show state memory areas used by vinyl for caches and write buffers.

| | |
|---|---|
| tnt_vinyl_memory_tuple_cache | Amount of memory in bytes currently used to store tuples (data) |
| tnt_vinyl_memory_level0 | «Level 0» (L0) memory area, bytes. L0 is the area that vinyl can use for in-memory storage of an LSM tree. By monitoring this metric, you can see when L0 is getting close to its maximum (tnt_vinyl_regulator_dump_watermark), at which time a dump will occur. You can expect L0 = 0 immediately after the dump operation is completed. |
| tnt_vinyl_memory_page_index | Amount of memory in bytes currently used to store indexes. If the metric value is close to *vinyl_memory*, this indicates that *vinyl_page_size* was chosen incorrectly. |
| tnt_vinyl_memory_bloom_filter | Amount of memory in bytes used by *bloom filters*. |

### Scheduler

The vinyl scheduler invokes the *regulator* and updates the related variables. This happens once per second.

| | |
|---|---|
| `tnt_vinyl_scheduler_tasks` | Number of scheduler dump/compaction tasks. The metric always has label `{status = <status_value>}`, where `<status_value>` can be one of the following:<br>• `inprogress` for currently running tasks<br>• `completed` for successfully completed tasks<br>• `failed` for tasks aborted due to errors. |
| `tnt_vinyl_scheduler_dump_time` | Total time in seconds spent by all worker threads performing dumps. |
| `tnt_vinyl_scheduler_dump_count` | Total number of dumps completed. |

### Event loop metrics

Event loop tx thread information:

| | |
|---|---|
| `tnt_ev_loop_time` | Event loop time (ms) |
| `tnt_ev_loop_prolog_time` | Event loop prolog time (ms) |
| `tnt_ev_loop_epilog_time` | Event loop epilog time (ms) |

### Synchro

Shows the current state of a synchronous replication.

| | |
|---|---|
| `tnt_synchro_queue_owner` | Instance ID of the current synchronous replication master. |
| `tnt_synchro_queue_term` | Current queue term. |
| `tnt_synchro_queue_len` | How many transactions are collecting confirmations now. |
| `tnt_synchro_queue_busy` | Whether the queue is processing any system entry (CONFIRM/ROLLBACK/PROMOTE/DEMOTE). |

### Election

Shows the current state of a replica set node in regards to leader election.

| | |
|---|---|
| `tnt_election_state` | election state (mode) of the node. When election is enabled, the node is writable only in the leader state. Possible values:<br>• 0 (`follower`) – all the non-leader nodes are called followers<br>• 1 (`candidate`) – the nodes that start a new election round are called candidates.<br>• 2 (`leader`) – the node that collected a quorum of votes becomes the leader |
| `tnt_election_vote` | ID of a node the current node votes for. If the value is 0, it means the node hasn't voted in the current term yet. |
| `tnt_election_leader` | Leader node ID in the current term. If the value is 0, it means the node doesn't know which node is the leader in the current term. |
| `tnt_election_term` | Current election term. |
| `tnt_election_leader_idle` | Time in seconds since the last interaction with the known leader. |

### Memtx

Memtx mvcc memory statistics. Transaction manager consists of two parts: - the transactions themselves (TXN section) - MVCC

### TXN

| | |
|---|---|
| `tnt_memtx_tnx_statements` are the transaction statements. | For example, the user started a transaction and made an action in it `space:replace{0, 1}`. Under the hood, this operation will turn into `statement` for the current transaction. This metric always has the label `{kind="..."}`, which has the following possible values:<br>• `total` The number of bytes that are allocated for the statements of all current transactions.<br>• `average` Average bytes used by transactions for statements (`txn.statements.total` bytes / number of open transactions).<br>• `max` The maximum number of bytes used by one the current transaction for statements. |
| `tnt_memtx_tnx_user` | In Tarantool C API there is a function `box_txn_alloc()`. By using this function user can allocate memory for the current transaction. This metric always has the label `{kind="..."}`, which has the following possible values:<br>• `total` Memory allocated by the `box_txn_alloc()` function on all current transactions.<br>• `average` Transaction average (total allocated bytes / number of all current transactions).<br>• `max` The maximum number of bytes allocated by `box_txn_alloc()` function per transaction. |
| `tnt_memtx_tnx_system` | There are internals: logs, savepoints. This metric always has the label `{kind="..."}`, which has the following possible values:<br>• `total` Memory allocated by internals on all current transactions.<br>• `average` Average allocated memory by internals (total memory / number of all current transactions).<br>• `max` The maximum number of bytes allocated by internals per transaction. |

### MVCC

`mvcc` is responsible for the isolation of transactions. It detects conflicts and makes sure that tuples that are no longer in the space, but read by some transaction (or can be read) have not been deleted.

| | |
|---|---|
| `tnt_memtx_mvcc_trackers` | Trackers that keep track of transaction reads. This metric always has the label `{kind="..."}`, which has the following possible values:<br>• `total` Trackers of all current transactions are allocated in total (in bytes).<br>• `average` Average for all current transactions (total memory bytes / number of transactions).<br>• `max` Maximum trackers allocated per transaction (in bytes). |
| `tnt_memtx_mvcc_conflicts` | Allocated in case of transaction conflicts. This metric always has the label `{kind="..."}`, which has the following possible values:<br>• `total` Bytes allocated for conflicts in total.<br>• `average` Average for all current transactions (total memory bytes / number of transactions).<br>• `max` Maximum bytes allocated for conflicts per transaction. |

#### Tuples

Saved tuples are divided into 3 categories: `used`, `read_view`, `tracking`.

Each category has two metrics: - `retained` tuples - they are no longer in the index, but MVCC does not allow them to be removed. - `stories` - MVCC is based on the story mechanism, almost every tuple has a story. This is a separate metric because even the tuples that are in the index can have a story. So `stories` and `retained` need to be measured separately.

| | |
|---|---|
| tnt_memtx_mvcc_tuples_used_stories | Tuple stories used by active read-write transactions. This metric always has the label {kind="..."}, which has the following possible values:<br>• count Number of used tuples / number of stories.<br>• total Amount of bytes used by stories used tuples. |
| tnt_memtx_mvcc_tuples_used_retained | Tuple stories used by active read-write transactions. But they are no longer in the index, but MVCC does not allow them to be removed. This metric always has the label {kind="..."}, which has the following possible values:<br>• count Number of retained used tuples / number of stories.<br>• total Amount of bytes used by retained used tuples. |
| tnt_memtx_mvcc_tuples_read_view_stories | Tuple stories used by active read-write transactions, but are used by read-only transactions (i.e. in read view). This metric always has the label {kind="..."}, which has the following possible values:<br>• count Number of read_view tuples / number of stories.<br>• total Amount of bytes used by stories read_view tuples. |
| tnt_memtx_mvcc_tuples_read_view_retained | Tuple stories used by active read-write transactions, but are used by read-only transactions (i.e. in read view). This tuples are no longer in the index, but MVCC does not allow them to be removed. This metric always has the label {kind="..."}, which has the following possible values:<br>• count Number of retained read_view tuples / number of stories.<br>• total Amount of bytes used by retained read_view tuples. |
| tnt_memtx_mvcc_tuples_tracking_stories | Tracking stories not directly used by any transactions, but are used by MVCC to track reads. This metric always has the label {kind="..."}, which has the following possible values:<br>• count Number of tracking tuples / number of tracking stories.<br>• total Amount of bytes used by stories tracking tuples. |
| tnt_memtx_mvcc_tuples_tracking_retained | Tracking stories not directly used by any transactions, but are used by MVCC to track reads. This tuples are no longer in the index, but MVCC does not allow them to be removed. This metric always has the label {kind="..."}, which has the following possible values:<br>• count Number of retained tracking tuples / number of stories.<br>• total Amount of bytes used by retained tracking tuples. |

### 6.17.5 API reference

#### Collectors

An application using the metrics module has 4 primitives, called **collectors**, at its disposal:

- *counter*
- *gauge*
- *histogram*
- *summary*

A collector represents one or more observations that change over time.

#### counter

A counter is a cumulative metric that denotes a single monotonically increasing counter. Its value might only increase or be reset to zero on restart. For example, you can use a counter to represent the number of requests served, tasks completed, or errors.

Don't use a counter to expose a value that can decrease. For example, don't use this metric to mark the number of currently running processes. Use a *gauge* type instead.

The design is based on the Prometheus counter.

`metrics.counter(`*name*[, *help*, *metainfo*]`)`
Register a new counter.

**Параметры**

- `name` (*string*) – collector name. Must be unique.
- `help` (*string*) – collector description.
- `metainfo` (*table*) – collector metainfo.

**Return** A counter object.

**Rtype** counter_obj

`object counter_obj`

`counter_obj:inc(`*num*, *label_pairs*`)`
Increment the observation for `label_pairs`. If `label_pairs` doesn't exist, the method creates it.

**Параметры**

- `num` (`number`) – increment value.
- `label_pairs` (*table*) – table containing label names as keys, label values as values. Note that both label names and values in `label_pairs` are treated as strings.

`counter_obj:collect()`

**Return** Array of `observation` objects for a given counter.

```
{
    label_pairs: table,          -- `label_pairs` key-value table
    timestamp: ctype<uint64_t>,  -- current system time (in microseconds)
    value: number,               -- current value
    metric_name: string,         -- collector
}
```

**Rtype** *table*

`counter_obj:remove(`*label_pairs*`)`
Remove the observation for `label_pairs`.

`counter_obj:reset(`*label_pairs*`)`
Set the observation for `label_pairs` to 0.

**Параметры**

- `label_pairs` (*table*) – table containing label names as keys, label values as values. Note that both label names and values in `label_pairs` are treated as strings.

### gauge

A gauge is a metric that denotes a single numerical value that can arbitrarily increase and decrease.

The gauge type is typically used for measured values like temperature or current memory usage. Also, it might be used for the values that can go up or down, for example, the number of concurrent requests.

The design is based on the Prometheus gauge.

`metrics.gauge(`*name*[, *help*, *metainfo*]`)`
Register a new gauge.

**Параметры**

- `name` (*string*) – collector name. Must be unique.
- `help` (*string*) – collector description.
- `metainfo` (*table*) – collector metainfo.

**Return** A gauge object.

**Rtype** gauge_obj

`object gauge_obj`

`gauge_obj:inc(`*num*, *label_pairs*`)`
Works like the `inc()` function of a *counter*.

`gauge_obj:dec(`*num*, *label_pairs*`)`
Works like `inc()`, but decrements the observation.

`gauge_obj:set(`*num*, *label_pairs*`)`
Sets the observation for `label_pairs` to `num`.

`gauge_obj:collect()`
Returns an array of `observation` objects for a given gauge. For the description of `observation`, see *counter_obj:collect()*.

`gauge_obj:remove(`*label_pairs*`)`
Works like the `remove()` function of a *counter*.

### histogram

A histogram metric is used to collect and analyze statistical data about the distribution of values within the application. Unlike metrics that only allow to track the average value or quantity of events, a histogram allows to see the distribution of values in detail and uncover hidden dependencies.

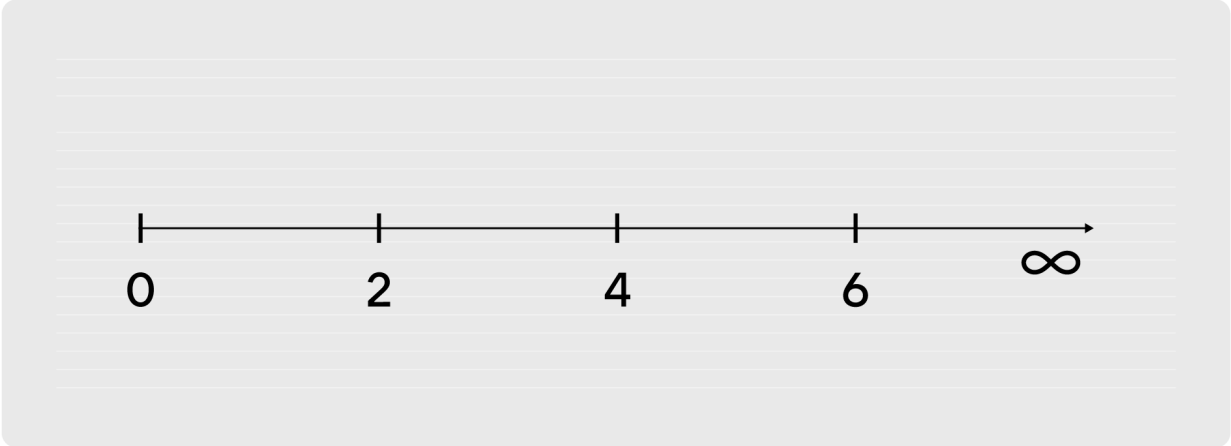
Suppose there are a lot of individual measurements that you don't want or can't store, and the aggregated information (the distribution of values across ranges) is enough to figure out the pattern. In this case, a histogram is used.

Each histogram provides several measurements:

- total count (`_count`)

- sum of measured values (`_sum`)
- distribution across buckets (`_bucket`)

Consider the following problem: you want to know how often the observed value is in the specific range (bucket).




For example, the observed values are 8, 7, 6, 8, 1, 7, 4, 8. Then, in the ranges:

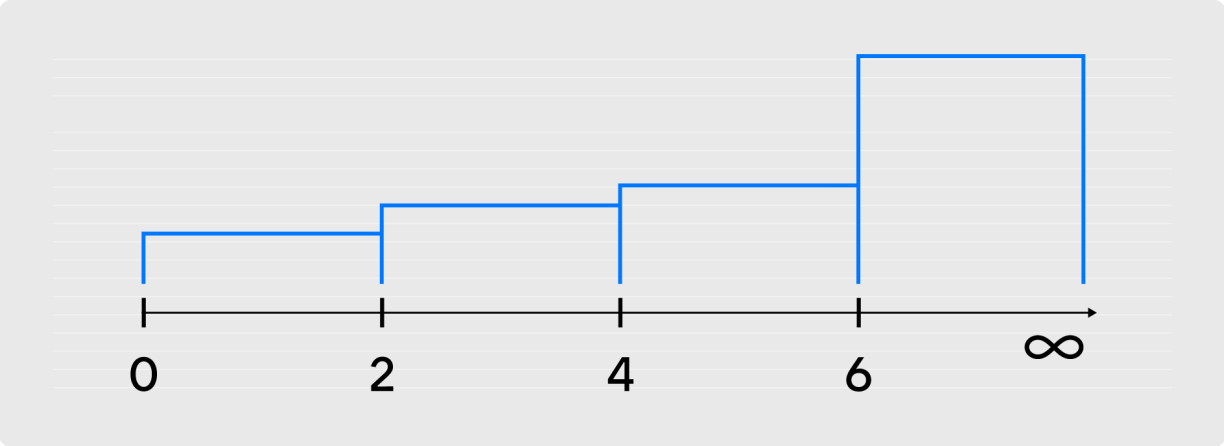
- In the interval [0, 2], there is 1 measurement.
- In the interval [0, 4], there are 2 measurements.
- In the interval [0, 6], there are 3 measurements.
- In the interval [0, infinity], there are 8 measurements (equal to the `histogram_demo_count` value).


```
{
    "label_pairs": {
       "le": 2,
       "alias": "my-tnt-app"
    },
    "timestamp": 1680174378390303,
    "metric_name": "histogram_demo_bucket",
    "value": 1
 },
 {
   "label_pairs": {
     "le": 4,
     "alias": "my-tnt-app"
   },
   "timestamp": 1680174378390303,
   "metric_name": "histogram_demo_bucket",
   "value": 2
 },
 {
   "label_pairs": {
     "le": 6,
     "alias": "my-tnt-app"
   },
   "timestamp": 1680174378390303,
```


(continues on next page)

(продолжение с предыдущей страницы)


```
  "metric_name": "histogram_demo_bucket",
  "value": 3
},
{
  "label_pairs": {
    "le": "inf",
    "alias": "my-tnt-app"
  },
  "timestamp": 1680174378390303,
  "metric_name": "histogram_demo_bucket",
  "value": 8
},
```


The metric also displays the count of measurements and their sum:


```
{
  "label_pairs": {
    "alias": "my-tnt-app"
  },
  "timestamp": 1680180929162484,
  "metric_name": "histogram_demo_count",
  "value": 8
},
{
  "label_pairs": {
    "alias": "my-tnt-app"
  },
  "timestamp": 1680180929162484,
  "metric_name": "histogram_demo_sum",
  "value": 49
},
```


The design is based on the Prometheus histogram.

`metrics.histogram`(*name*[, *help*, *buckets*, *metainfo*])

Register a new histogram.

**Параметры**

- `name` (*string*) – collector name. Must be unique.
- `help` (*string*) – collector description.
- `buckets` (*table*) – histogram buckets (an array of sorted positive numbers). The infinity bucket (`INF`) is appended automatically. Default: `{.005, .01, .025, .05, .075, .1, .25, .5, .75, 1.0, 2.5, 5.0, 7.5, 10.0, INF}`.
- `metainfo` (*table*) – collector metainfo.

**Return** A histogram object.

**Rtype** histogram_obj

---

**Примечание:** A histogram is basically a set of collectors:

- `name .. "_sum"` – a counter holding the sum of added observations.
- `name .. "_count"` – a counter holding the number of added observations.
- `name .. "_bucket"` – a counter holding all bucket sizes under the label `le` (less or equal). To access a specific bucket – `x` (where `x` is a number), specify the value `x` for the label `le`.

---

`object histogram_obj`

`histogram_obj:observe(`*num*, *label_pairs*`)`
Record a new value in a histogram. This increments all bucket sizes under the labels `le` >= `num` and the labels that match `label_pairs`.

**Параметры**

- `num` (`number`) – value to put in the histogram.
- `label_pairs` (*table*) – table containing label names as keys, label values as values. All internal counters that have these labels specified observe new counter values. Note that both label names and values in `label_pairs` are treated as strings.

`histogram_obj:collect()`
Return a concatenation of `counter_obj:collect()` across all internal counters of `histogram_obj`. For the description of `observation`, see *counter_obj:collect()*.

`histogram_obj:remove(`*label_pairs*`)`
Works like the `remove()` function of a *counter*.

### summary

A summary metric is used to collect statistical data about the distribution of values within the application.

Each summary provides several measurements:

- total count of measurements
- sum of measured values
- values at specific quantiles

Similar to histograms, a summary also operates with value ranges. However, unlike histograms, it uses quantiles (defined by a number between 0 and 1) for this purpose. In this case, it is not required to define fixed boundaries. For summary type, the ranges depend on the measured values and the number of measurements.

Suppose the example series of measurements are sorted in ascending order: 1, 4, 6, 7, 7, 8, 8, 8.

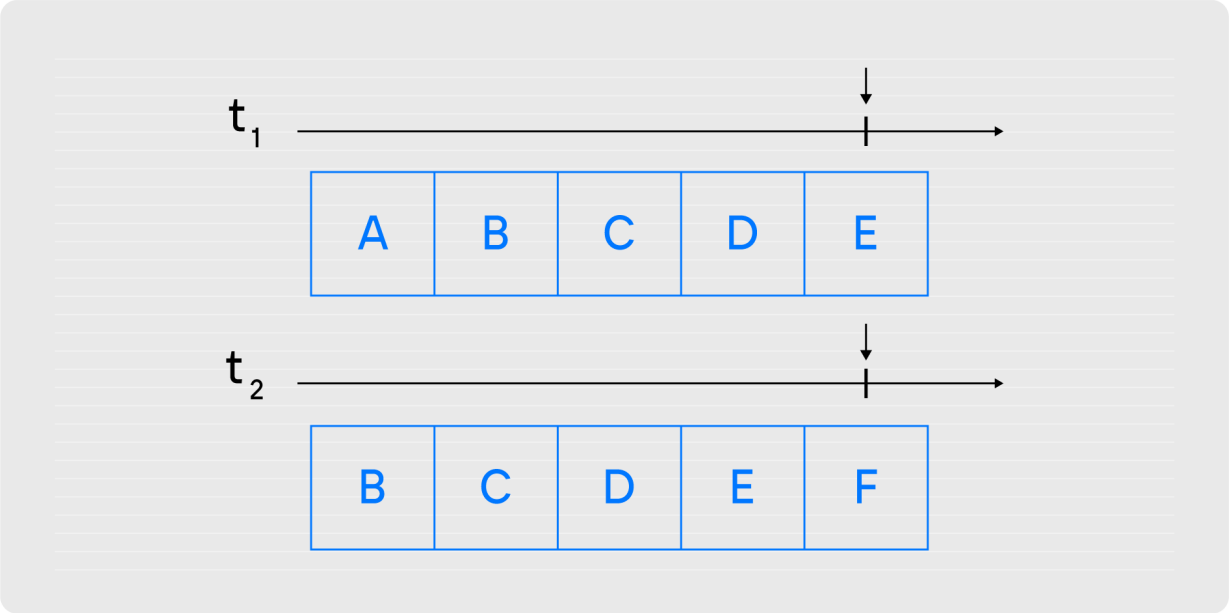
Quantiles:

- Quantile 0 is the value of the first, minimum element. In this example, it's 1.
- Quantile 1 is the value of the last, maximum element. In this example, it's 8.
- Quantile 0.5 is the value of the median element. In this example, it's 7. This means that the smaller half of the measurements is a range of values from 1 to 7. The larger one is a range of values from 7 to 8.

Note that calculating quantiles requires resources, so it makes sense to calculate no more than one, for example: 0.95 – the majority of measurements.

With a large number of measurements per second, a significant amount of memory is required to store them all. The array is compressed to reduce memory consumption. The degree of compression is determined by an acceptable error rate. In application, error rates mostly from 1% to 10%. This means that a quantile of 0.50 with a 10% error from the example above returns a value in the range of 6.65. . . 7.35 instead of 7.

Additionally, a summary metric doesn't store values for the whole application's lifetime. This metric uses a sliding window divided into sections (buckets) where measurements are stored.




Note that buckets in histograms and buckets in quantiles within summaries have different meanings.

```
local summary_demo = metrics.summary(
    'summary_demo', -- metric name
    'Summary demo', -- description
    {
       [0.5] = 0.01, -- quantile 0.50 with 1% error
       [0.95] = 0.01, -- quantile 0.95 with 1% error
       [0.99] = 0.01, -- quantile 0.99 with 1% error
    },
    {
       max_age_time = 60, -- duration of each bucket in seconds
       age_buckets_count = 5 -- total number of buckets in the sliding window
                             -- window duration = max_age_time * age_buckets_count␣
↪seconds, or in
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                -- this case = 5 minutes
    }
)
```

The metric like in the example above returns the following measurements for the specified quantiles:

```
{
   "label_pairs": {
      "quantile": 0.5,
      "alias": "my-tnt-app"
   },
   "timestamp": 1680180929162484,
   "metric_name": "summary_demo",
   "value": 7
  },
  {
    "label_pairs": {
      "quantile": 0.95,
      "alias": "my-tnt-app"
    },
    "timestamp": 1680180929162484,
    "metric_name": "summary_demo",
    "value": 8
  },
  {
    "label_pairs": {
      "quantile": 0.99,
      "alias": "my-tnt-app"
    },
    "timestamp": 1680180929162484,
    "metric_name": "summary_demo",
    "value": 8
  },
```

Also, the metric exposes the count of measurements and the sum of observations:

```
{
  "label_pairs": {
    "alias": "my-tnt-app"
  },
  "timestamp": 1680180929162484,
  "metric_name": "summary_demo_count",
  "value": 8
},
{
  "label_pairs": {
    "alias": "my-tnt-app"
  },
  "timestamp": 1680180929162484,
  "metric_name": "summary_demo_sum",
  "value": 49
},
```

The design is based on the Prometheus summary.

`metrics.summary`(*name*[, *help*, *objectives*, *params*, *metainfo*])

Register a new summary. Quantile computation is based on the «Effective computation of biased quantiles over data streams» algorithm.

**Параметры**

- `name` (*string*) – collector name. Must be unique.
- `help` (*string*) – collector description.
- `objectives` (*table*) – a list of «targeted» -quantiles in the `{quantile = error, ... }` form. Example: `{[0.5]=0.01, [0.9]=0.01, [0.99]=0.01}`. The targeted -quantile is specified in the form of a -quantile and the tolerated error. For example, `{[0.5] = 0.1}` means that the median (= 50th percentile) is to be returned with a 10-percent error. Note that percentiles and quantiles are the same concept, except that percentiles are expressed as percentages. The -quantile must be in the interval `[0, 1]`. A lower tolerated error for a -quantile results in higher memory and CPU usage during summary calculation.
- `params` (*table*) – table of the summary parameters used to configuring the sliding time window. This window consists of several buckets to store observations. New observations are added to each bucket. After a time period, the head bucket (from which observations are collected) is reset, and the next bucket becomes the new head. This way, each bucket stores observations for `max_age_time * age_buckets_count` seconds before it is reset. `max_age_time` sets the duration of each bucket's lifetime – that is, how many seconds the observations are kept before they are discarded. `age_buckets_count` sets the number of buckets in the sliding time window. This variable determines the number of buckets used to exclude observations older than `max_age_time` from the summary. The value is a trade-off between resources (memory and CPU for maintaining the bucket) and how smooth the time window moves. Default value: `{max_age_time = math.huge, age_buckets_count = 1}`.
- `metainfo` (*table*) – collector metainfo.

**Return** A summary object.

**Rtype** summary_obj

---

**Примечание:** A summary represents a set of collectors:

- `name .. "_sum"` – a counter holding the sum of added observations.
- `name .. "_count"` – a counter holding the number of added observations.
- `name` holds all the quantiles under observation that find themselves under the label `quantile` (less or equal). To access bucket `x` (where `x` is a number), specify the value `x` for the label `quantile`.

---

`object summary_obj`

`summary_obj:observe`(*num*, *label_pairs*)

Record a new value in a summary.

**Параметры**

- `num` (`number`) – value to put in the data stream.
- `label_pairs` (*table*) – a table containing label names as keys, label values as values. All internal counters that have these labels specified observe new

counter values. You can't add the `"quantile"` label to a summary. It is added automatically. If `max_age_time` and `age_buckets_count` are set, the observed value is added to each bucket. Note that both label names and values in `label_pairs` are treated as strings.

`summary_obj:collect()`
Return a concatenation of `counter_obj:collect()` across all internal counters of `summary_obj`. For the description of `observation`, see *counter_obj:collect()*. If `max_age_time` and `age_buckets_count` are set, quantile observations are collected only from the head bucket in the sliding time window, not from every bucket. If no observations were recorded, the method will return `NaN` in the values.

`summary_obj:remove(`*label_pairs*`)`
Works like the `remove()` function of a *counter*.

### Labels

All collectors support providing `label_pairs` on data modification. A label is a piece of metainfo that you associate with a metric in the key-value format. For details, see tags in Graphite and labels in Prometheus.

Labels are used to differentiate between the characteristics of a thing being measured. For example, in a metric associated with the total number of HTTP requests, you can represent methods and statuses as label pairs:

```
http_requests_total_counter:inc(1, {method = 'POST', status = '200'})
```

You don't have to predefine labels in advance.

With labels, you can extract new time series (visualize their graphs) by specifying conditions with regard to label values. The example above allows extracting the following time series:

1. The total number of requests over time with `method = "POST"` (and any status).
2. The total number of requests over time with `status = 500` (and any method).

You can also set global labels by calling `metrics.set_global_labels({ label = value, ...})`.

### Metrics functions

`metrics.cfg(`[*config*]`)`
Entrypoint to setup the module. Since 0.17.0.

**Параметры**

- `config` (*table*) – module configuration options:
  - `cfg.include` (string/table, default `'all'`): `'all'` to enable all supported default metrics, `'none'` to disable all default metrics, table with names of the default metrics to enable a specific set of metrics.
  - `cfg.exclude` (table, default `{}`): table containing the names of the default metrics that you want to disable. Has higher priority than `cfg.include`.
  - `cfg.labels` (table, default `{}`): table containing label names as string keys, label values as values.

You can work with `metrics.cfg` as a table to read values, but you must call `metrics.cfg{}` as a function to update them.

Supported default metric names (for `cfg.include` and `cfg.exclude` tables):

- `all` (metasection including all metrics)
- `network`
- `operations`
- `system`
- `replicas`
- `info`
- `slab`
- `runtime`
- `memory`
- `spaces`
- `fibers`
- `cpu`
- `vinyl`
- `memtx`
- `luajit`
- `cartridge_issues`
- `cartridge_failover`
- `clock`
- `event_loop`

See *metrics reference* for details. All metric collectors from the collection have `metainfo.default = true`.

`cfg.labels` are the global labels to be added to every observation.

Global labels are applied only to metric collection. They have no effect on how observations are stored.

Global labels can be changed on the fly.

`label_pairs` from observation objects have priority over global labels. If you pass `label_pairs` to an observation method with the same key as some global label, the method argument value will be used.

Note that both label names and values in `label_pairs` are treated as strings.

`metrics.enable_default_metrics(`[*include*, *exclude*]`)`
Same as `metrics.cfg{include=include, exclude=exclude}`, but `include={}` is treated as `include='all'` for backward compatibility.

`metrics.set_global_labels(`*label_pairs*`)`
Same as `metrics.cfg{labels=label_pairs}`.

`metrics.collect(`[*opts*]`)`
Collect observations from each collector.

**Параметры**

- `opts` (*table*) – table of collect options:
  - `invoke_callbacks` – if `true`, `invoke_callbacks()` is triggered before actual collect.

– `default_only` – if `true`, observations contain only default metrics (`metainfo.default = true`).

`object registry`

`registry:unregister(`*collector*`)`
Remove a collector from the registry.

**Параметры**

- `collector` (`collector_obj`) – the collector to be removed.

**Example:**

```
local collector = metrics.gauge('some-gauge')

-- after a while, we don't need it anymore

metrics.registry:unregister(collector)
```

`registry:find(`*kind*, *name*`)`
Find a collector in the registry.

**Параметры**

- `kind` (*string*) – collector kind (`counter`, `gauge`, `histogram`, or `summary`).
- `name` (*string*) – collector name.

**Return** A collector object or `nil`.

**Rtype** collector_obj

**Example:**

```
local collector = metrics.gauge('some-gauge')

collector = metrics.registry:find('gauge', 'some-gauge')
```

`metrics.register_callback(`*callback*`)`
Register a function named `callback`, which will be called right before metric collection on plugin export.

**Параметры**

- `callback` (`function`) – a function that takes no parameters.

This method is most often used for gauge metrics updates.

**Example:**

```
metrics.register_callback(function()
    local cpu_metrics = require('metrics.psutils.cpu')
    cpu_metrics.update()
end)
```

`metrics.unregister_callback(`*callback*`)`
Unregister a function named `callback` that is called right before metric collection on plugin export.

**Параметры**

- `callback` (`function`) – a function that takes no parameters.

**Example:**

```
local cpu_callback = function()
    local cpu_metrics = require('metrics.psutils.cpu')
    cpu_metrics.update()
end

metrics.register_callback(cpu_callback)

-- after a while, we don't need that callback function anymore

metrics.unregister_callback(cpu_callback)
```

`metrics.invoke_callbacks()`
Invoke all registered callbacks. Has to be called before each `collect()`. (Since version **0.16.0**, you may use `collect{invoke_callbacks = true}` instead.) If you're using one of the default exporters, `invoke_callbacks()` will be called by the exporter.

### Metrics role API

Below are the functions that you can call with `metrics = require('cartridge.roles.metrics')` specified in your `init.lua`.

`metrics.set_export(`*export*`)`

**Параметры**

- `export` (*table*) – a table containing paths and formats of the exported metrics.

Configure the endpoints of the metrics role:

```
local metrics = require('cartridge.roles.metrics')
metrics.set_export({
    {
        path = '/path_for_json_metrics',
        format = 'json'
    },
    {
        path = '/path_for_prometheus_metrics',
        format = 'prometheus'
    },
    {
        path = '/health',
        format = 'health'
    }
})
```

You can add several entry points of the same format but with different paths, for example:

```
metrics.set_export({
    {
        path = '/path_for_json_metrics',
        format = 'json'
    },
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    {
        path = '/another_path_for_json_metrics',
        format = 'json'
    },
})
```

metrics.set_default_labels(*label_pairs*)
Add default global labels. Note that both label names and values in `label_pairs` are treated as strings.

**Параметры**

- `label_pairs` (*table*) – Table containing label names as string keys, label values as values.

```
local metrics = require('cartridge.roles.metrics')
metrics.set_default_labels({ ['my-custom-label'] = 'label-value' })
```

### Collecting HTTP request latency statistics

The `metrics` module provides middleware for monitoring HTTP (set by the http module) latency statistics. The latency collector observes both latency information and the number of invocations. The metrics collected by HTTP middleware are separated by a set of labels:

- route (`path`)
- method (`method`)
- HTTP status code (`status`)

For each route that you want to track, you must specify the middleware explicitly. The middleware does not cover the 404 errors.

metrics.http_middleware.configure_default_collector(*type_name*, *name*, *help*)
Register a collector for the middleware and set it as default.

**Параметры**

- `type_name` (*string*) – collector type: `histogram` or `summary`. The default is `histogram`.
- `name` (*string*) – collector name. The default is `http_server_request_latency`.
- `help` (*string*) – collector description. The default is `HTTP Server Request Latency`.

**Possible errors:**

- A collector with the same type and name already exists in the registry.

metrics.http_middleware.build_default_collector(*type_name*, *name*[, *help*])
Register and return a collector for the middleware.

**Параметры**

- `type_name` (*string*) – collector type: `histogram` or `summary`. The default is `histogram`.
- `name` (*string*) – collector name. The default is `http_server_request_latency`.

- `help` (*string*) – collector description. The default is `HTTP Server Request Latency`.

**Return** A collector object.

**Possible errors:**

- A collector with the same type and name already exists in the registry.

`metrics.http_middleware.set_default_collector(`*collector*`)`
Set the default collector.

**Параметры**

- `collector` – middleware collector object.

`metrics.http_middleware.get_default_collector()`
Return the default collector. If the default collector hasn't been set yet, register it (with default `http_middleware.build_default_collector(...)` parameters) and set it as default.

**Return** A collector object.

`metrics.http_middleware.v1(`*handler*, *collector*`)`
Latency measuring wrap-up for the HTTP ver. `1.x.x` handler. Returns a wrapped handler.

**Параметры**

- `handler` (`function`) – handler function.
- `collector` – middleware collector object. If not set, the default collector is used (like in `http_middleware.get_default_collector()`).

**Usage:** `httpd:route(route, http_middleware.v1(request_handler, collector))`

**Example:**

```
#!/usr/bin/env tarantool
package.path = package.path .. ";../?.lua"

local json = require('json')
local fiber = require('fiber')
local metrics = require('metrics')
local log = require('log')
local http_middleware = metrics.http_middleware

-- Configure HTTP routing
local ip = '127.0.0.1'
local port = 12345
local httpd = require('http.server').new(ip, port) -- HTTP ver. 1.x.x
local route = { path = '/path', method = 'POST' }

-- Route handler
local handler = function(req)
    for _ = 1, 10 do
        fiber.sleep(0.1)
    end

    return { status = 200, body = req.body }
end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Configure summary latency collector
local collector = http_middleware.build_default_collector('summary')

-- Set route handler with summary latency collection
httpd:route(route, http_middleware.v1(handler, collector))
-- Start HTTP routing
httpd:start()

-- Set HTTP client, make some request
local http_client = require("http.client") -- HTTP ver. 1.x.x
http_client.post('http://' .. ip .. ':' .. port .. route.path, json.encode({ body =
↪'text' }))

-- Collect the metrics
log.info(metrics.collect())
--[[

- label_pairs:
    path: /path
    method: POST
    status: 200
  timestamp: 1588951616500768
  value: 1
  metric_name: path_latency_count

- label_pairs:
    path: /path
    method: POST
    status: 200
  timestamp: 1588951616500768
  value: 1.0240110000595
   metric_name: path_latency_sum

 - label_pairs:
     path: /path
     method: POST
     status: 200
     quantile: 0.5
   timestamp: 1588951616500768
   value: 1.0240110000595
   metric_name: path_latency

 - label_pairs:
     path: /path
     method: POST
     status: 200
     quantile: 0.9
   timestamp: 1588951616500768
   value: 1.0240110000595
   metric_name: path_latency

 - label_pairs:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    path: /path
    method: POST
    status: 200
    quantile: 0.99
  timestamp: 1588951616500768
  value: 1.0240110000595
  metric_name: path_latency

--]]

-- Exit event loop
os.exit()
```

### CPU usage metrics

CPU metrics work only on Linux. See the *metrics reference* for details.

To enable CPU metrics, first register a callback function:

```
local metrics = require('metrics')

local cpu_callback = function()
    local cpu_metrics = require('metrics.psutils.cpu')
    cpu_metrics.update()
end

metrics.register_callback(cpu_callback)
```

**Collected metrics example:**

```
# HELP tnt_cpu_time Host CPU time
# TYPE tnt_cpu_time gauge
tnt_cpu_time 15006759
# HELP tnt_cpu_thread Tarantool thread cpu time
# TYPE tnt_cpu_thread gauge
tnt_cpu_thread{thread_name="coio",file_name="init.lua",thread_pid="699",kind="system"}␣
↪160
tnt_cpu_thread{thread_name="tarantool",file_name="init.lua",thread_pid="1",kind="user"}␣
↪949
tnt_cpu_thread{thread_name="tarantool",file_name="init.lua",thread_pid="1",kind="system"}
↪ 920
tnt_cpu_thread{thread_name="coio",file_name="init.lua",thread_pid="11",kind="user"} 79
tnt_cpu_thread{thread_name="coio",file_name="init.lua",thread_pid="699",kind="user"} 44
tnt_cpu_thread{thread_name="coio",file_name="init.lua",thread_pid="11",kind="system"} 294
```

**Prometheus query aggregated by thread name:**

```
sum by (thread_name) (idelta(tnt_cpu_thread[$__interval]))
  / scalar(idelta(tnt_cpu_total[$__interval]) / tnt_cpu_count)
```

All psutils metric collectors have `metainfo.default = true`.

To clear CPU metrics when you don't need them anymore, remove the callback and clear the collectors with a method:

```
metrics.unregister_callback(cpu_callback)
cpu_metrics.clear()
```

### Examples

Below are some examples of using metric primitives.

Notice that this usage is independent of export plugins such as Prometheus, Graphite, etc. For documentation on how to use the plugins, see the *Metrics plugins* section.

**Using counters:**

```
local metrics = require('metrics')

-- create a counter
local http_requests_total_counter = metrics.counter('http_requests_total')

-- somewhere in the HTTP requests middleware:
http_requests_total_counter:inc(1, {method = 'GET'})
```

**Using gauges:**

```
local metrics = require('metrics')

-- create a gauge
local cpu_usage_gauge = metrics.gauge('cpu_usage', 'CPU usage')

-- register a lazy gauge value update
-- this will be called whenever export is invoked in any plugins
metrics.register_callback(function()
    local current_cpu_usage = some_cpu_collect_function()
    cpu_usage_gauge:set(current_cpu_usage, {app = 'tarantool'})
end)
```

**Using histograms:**

```
local metrics = require('metrics')
local fiber = require('fiber')
-- create a histogram
local http_requests_latency_hist = metrics.histogram(
    'http_requests_latency', 'HTTP requests total', {2, 4, 6})

-- somewhere in the HTTP request middleware:

local t0 = fiber.clock()
observable_function()
local t1 = fiber.clock()

local latency = t1 - t0
http_requests_latency_hist:observe(latency)
```

**Using summaries:**

```
local metrics = require('metrics')
local fiber = require('fiber')

-- create a summary with a window of 5 age buckets and a bucket lifetime of 60 s
local http_requests_latency = metrics.summary(
    'http_requests_latency', 'HTTP requests total',
    {[0.5]=0.01, [0.9]=0.01, [0.99]=0.01},
    {max_age_time = 60, age_buckets_count = 5}
)

-- somewhere in the HTTP requests middleware:
local t0 = fiber.clock()
observable_function()
local t1 = fiber.clock()

local latency = t1 - t0
http_requests_latency:observe(latency)
```

### 6.17.6 Metrics plugins

Plugins let you collect metrics through a unified interface without worrying about the way metrics export works. If you want to use another DB to store metrics data, you can enable an appropriate export plugin just by changing one line of code.

#### Available plugins

#### Prometheus

#### Usage

Import the plugin:

```
local prometheus = require('metrics.plugins.prometheus')
```

Then use the `prometheus.collect_http()` function, which returns:

```
{
    status = 200,
    headers = <headers>,
    body = <body>,
}
```

See the Prometheus exposition format for details on `<body>` and `<headers>`.

With Tarantool http.server, use this plugin as follows:

```
local httpd = require('http.server').new(...)
...
httpd:route( { path = '/metrics' }, prometheus.collect_http)
```

### Sample settings

- For Tarantool:

```
metrics = require('metrics')
metrics.cfg{}
prometheus = require('metrics.plugins.prometheus')
httpd = require('http.server').new('0.0.0.0', 8080)
httpd:route( { path = '/metrics' }, prometheus.collect_http)
httpd:start()
```

- For Tarantool Cartridge:

```
cartridge = require('cartridge')
httpd = cartridge.service_get('httpd')
metrics = require('metrics')
metrics.cfg{}
prometheus = require('metrics.plugins.prometheus')
httpd:route( { path = '/metrics' }, prometheus.collect_http)
```

## Graphite

### Usage

Import the plugin:

```
local graphite = require('metrics.plugins.graphite')
```

To start automatically exporting the current values of all `metrics.{counter,gauge,histogram}`, call the following function:

`metrics.plugins.graphite.init(`*options*`)`

**Параметры**

- `options` (*table*) – possible options:
  - `prefix` (string): metrics prefix (`'tarantool'` by default)
  - `host` (string): Graphite server host (`'127.0.0.1'` by default)
  - `port` (number): Graphite server port (2003 by default)
  - `send_interval` (number): metrics collection interval in seconds (2 by default)

This function creates a background fiber that periodically sends all metrics to a remote Graphite server.

Exported metric names are formatted as follows: `<prefix>.<metric_name>`.

### JSON

#### Usage

Import the plugin:

```
local json_metrics = require('metrics.plugins.json')
```

metrics.plugins.json.export()

**Return** the following structure

```
[
    {
        "name": "<name>",
        "label_pairs": {
            "<name>": "<value>",
            "...": "..."
            },
        "timestamp": "<number>",
        "value": "<value>"
    },
    "..."
]
```

**Rtype** *string*

---

**Важно:** The values can also be `+-math.huge` and `math.huge * 0`. In such case:

- `math.huge` is serialized to `"inf"`
- `-math.huge` is serialized to `"-inf"`
- `math.huge * 0` is serialized to `"nan"`.

---

**Example**

```
[
    {
        "label_pairs": {
            "type": "nan"
        },
        "timestamp": 1559211080514607,
        "metric_name": "test_nan",
        "value": "nan"
    },
    {
        "label_pairs": {
            "type": "-inf"
        },
        "timestamp": 1559211080514607,
        "metric_name": "test_inf",
        "value": "-inf"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    },
    {
        "label_pairs": {
            "type": "inf"
        },
        "timestamp": 1559211080514607,
        "metric_name": "test_inf",
        "value": "inf"
    }
]
```

Use the JSON plugin with Tarantool `http.server` as follows:

```
local httpd = require('http.server').new(...)
...
httpd:route({
        method = 'GET',
        path = '/metrics',
        public = true,
    },
    function(req)
        return req:render({
            text = json_exporter.export()
        })
    end
)
```

### Plugin-specific API

Use the following methods **only when developing a new plugin**.

`metrics.invoke_callbacks()`
Invoke a function registered via `metrics.register_callback(<callback>)`. Used in exporters.

`metrics.collectors()`
List all collectors in the registry. Designed to be used in exporters.

**Return** A list of created collectors.

`object collector_object`

`collector_object:collect()`

**Примечание:** You'll probably want to use `metrics.collectors()` instead.

Equivalent to:

```
for _, c in pairs(metrics.collectors()) do
    for _, obs in ipairs(c:collect()) do
        ...  -- handle observation
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    end
end
```

**Return** A concatenation of `observation` objects across all created collectors.

```
{
    label_pairs: table,          -- `label_pairs` key-value table
    timestamp: ctype<uint64_t>, -- current system time (in
↪microseconds)
    value: number,               -- current value
    metric_name: string,         -- collector
}
```

**Rtype** *table*

### Creating custom plugins

Include the following in your main export function:

```
-- Invoke all callbacks registered via `metrics.register_callback(<callback-function>)`
metrics.invoke_callbacks()

-- Loop over collectors
for _, c in pairs(metrics.collectors()) do
    ...

    -- Loop over instant observations in the collector
    for _, obs in pairs(c:collect()) do
        -- Export observation `obs`
        ...
    end

end
```

## 6.17.7 Grafana dashboard

Tarantool Grafana dashboard is available as part of Grafana Official & community built dashboards. There's a version for Prometheus data source and one for InfluxDB data source. There are also separate dashboards for TDG applications: for Prometheus data source and for InfluxDB data source. Tarantool Grafana dashboard is a ready for import template with basic memory, space operations, and HTTP load panels, based on default metrics package functionality.

Dashboard requires using `metrics` **0.15.0** or newer for complete experience; `'alias'` *global label* must be set on each instance to properly display panels (e.g. provided with `cartridge.roles.metrics` role).

To support CRUD statistics, install `CRUD` **0.11.1** or newer. Call `crud.cfg` on router to enable CRUD statistics collect with latency quantiles.

```
crud.cfg{
    stats = true,
    stats_driver='metrics',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    stats_quantiles=true
}
```

To support expiriationd statistics, install `expiriationd` **1.2.0** or newer. Call `expiriationd.cfg` on instance to enable statistics export.

```
expiriationd.cfg{metrics = true}
```

### Prepare a monitoring stack

Since there are Prometheus and InfluxDB data source Grafana dashboards, you can use

- Telegraf as a server agent for collecting metrics, InfluxDB as a time series database for storing metrics, and Grafana as a visualization platform; or
- Prometheus as both a server agent for collecting metrics and a time series database for storing metrics, and Grafana as a visualization platform.

For issues concerning setting up Prometheus, Telegraf, InfluxDB, or Grafana instances please refer to the corresponding project's documentation.

### Collect metrics with server agents

To collect metrics for Prometheus, first set up metrics output with `prometheus` format. You can use cartridge.roles.metrics configuration or set up the *Prometheus output plugin* manually. To start collecting metrics, add a job to Prometheus configuration with each Tarantool instance URI as a target and metrics path as it was configured on Tarantool instances:

```
scrape_configs:
  - job_name: tarantool
    static_configs:
      - targets:
        - "example_project:8081"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        - "example_project:8082"
        - "example_project:8083"
    metrics_path: "/metrics/prometheus"
```

To collect metrics for InfluxDB, use the Telegraf agent. First off, configure Tarantool metrics output in `json` format with cartridge.roles.metrics configuration or corresponding *JSON output plugin*. To start collecting metrics, add http input to Telegraf configuration including each Tarantool instance metrics URL:

```
[[inputs.http]]
    urls = [
        "http://example_project:8081/metrics/json",
        "http://example_project:8082/metrics/json",
        "http://example_project:8083/metrics/json"
    ]
    timeout = "30s"
    tag_keys = [
        "metric_name",
        "label_pairs_alias",
        "label_pairs_quantile",
        "label_pairs_path",
        "label_pairs_method",
        "label_pairs_status",
        "label_pairs_operation",
        "label_pairs_level",
        "label_pairs_id",
        "label_pairs_engine",
        "label_pairs_name",
        "label_pairs_index_name",
        "label_pairs_delta",
        "label_pairs_stream",
        "label_pairs_thread",
        "label_pairs_kind"
    ]
    insecure_skip_verify = true
    interval = "10s"
    data_format = "json"
    name_prefix = "tarantool_"
    fieldpass = ["value"]
```

Be sure to include each label key as `label_pairs_<key>` so it will be extracted with plugin. For example, if you use `{ state = 'ready' }` labels somewhere in metric collectors, add `label_pairs_state` tag key.

For TDG dashboard, please use

```
[[inputs.http]]
    urls = [
        "http://example_tdg_project:8081/metrics/json",
        "http://example_tdg_project:8082/metrics/json",
        "http://example_tdg_project:8083/metrics/json"
    ]
    timeout = "30s"
    tag_keys = [
        "metric_name",
```

(continues on next page)

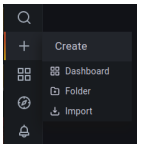
(продолжение с предыдущей страницы)

```
        "label_pairs_alias",
        "label_pairs_quantile",
        "label_pairs_path",
        "label_pairs_method",
        "label_pairs_status",
        "label_pairs_operation",
        "label_pairs_level",
        "label_pairs_id",
        "label_pairs_engine",
        "label_pairs_name",
        "label_pairs_index_name",
        "label_pairs_delta",
        "label_pairs_stream",
        "label_pairs_thread",
        "label_pairs_type",
        "label_pairs_connector_name",
        "label_pairs_broker_name",
        "label_pairs_topic",
        "label_pairs_request",
        "label_pairs_kind",
        "label_pairs_thread_name",
        "label_pairs_type_name",
        "label_pairs_operation_name",
        "label_pairs_schema",
        "label_pairs_entity",
        "label_pairs_status_code"
    ]
    insecure_skip_verify = true
    interval = "10s"
    data_format = "json"
    name_prefix = "tarantool_"
    fieldpass = ["value"]
```

If you connect Telegraf instance to InfluxDB storage, metrics will be stored with `"<name_prefix>http"` measurement (`"tarantool_http"` in our example).

### Import the dashboard

Open Grafana import menu.



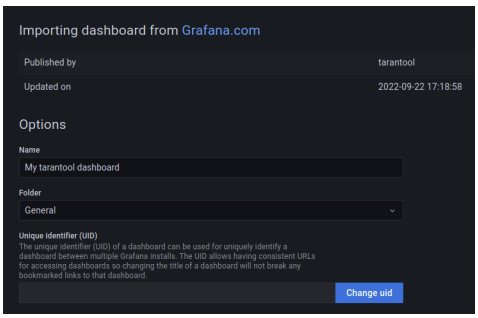

To import a specific dashboard, choose one of the following options:

- paste the dashboard id (`12567` for InfluxDB dashboard, `13054` for Prometheus dashboard, `16405` for InfluxDB TDG dashboard, `16406` for Prometheus TDG dashboard), or
- paste a link to the dashboard ( https://grafana.com/grafana/dashboards/12567 for InfluxDB dashboard, https://grafana.com/grafana/dashboards/13054 for Prometheus dashboard, https:

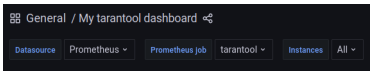
//grafana.com/grafana/dashboards/16405 for InfluxDB TDG dashboard, https://grafana.com/grafana/dashboards/16406 for Prometheus TDG dashboard), or

- paste the dashboard JSON file contents, or
- upload the dashboard JSON file.

Set dashboard name, folder and uid (if needed).




You can choose datasource and datasource variables after import.




#### Troubleshooting

If there are no data on the graphs, make sure that you picked datasource and job/measurement correctly.

If there are no data on the graphs, make sure that you have `info` group of Tarantool metrics (in particular, `tnt_info_uptime`).

If some Prometheus graphs show no data because of `parse error: missing unit character in duration`, ensure that you use Grafana 7.2 or newer.

If some Prometheus graphs display `parse error: bad duration syntax "1m0"` or similar error, you need to update your Prometheus version. See grafana/grafana#44542 for more details.

### 6.17.8 Alerting

You can set up alerts on metrics to get a notification when something went wrong. We will use Prometheus alert rules as an example here. You can get full `alerts.yml` file at tarantool/grafana-dashboard GitHub repo.

#### Tarantool metrics

You can use internal Tarantool metrics to monitor detailed RAM consumption, replication state, database engine status, track business logic issues (like HTTP 4xx and 5xx responses or low request rate) and external modules statistics (like `CRUD` errors or Cartridge issues). Evaluation timeouts, severity levels and thresholds (especially ones for business logic) are placed here for the sake of example: you may want to increase or decrease them for your application. Also, don't forget to set sane rate time ranges based on your Prometheus configuration.

### Lua memory

The Lua memory is limited to 2 GB per instance. Monitoring `tnt_info_memory_lua` metric may prevent memory overflow and detect the presence of bad Lua code practices.

```
- alert: HighLuaMemoryWarning
  expr: tnt_info_memory_lua >= (512 * 1024 * 1024)
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') Lua runtime warning"
    description: "'{{ $labels.alias }}' instance of job '{{ $labels.job }}' uses too␣
↪much Lua memory
      and may hit threshold soon."

- alert: HighLuaMemoryAlert
  expr: tnt_info_memory_lua >= (1024 * 1024 * 1024)
  for: 1m
  labels:
    severity: page
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') Lua runtime alert"
    description: "'{{ $labels.alias }}' instance of job '{{ $labels.job }}' uses too␣
↪much Lua memory
      and likely to hit threshold soon."
```

### Memtx arena memory

By monitoring *slab allocation statistics* you can see how many free RAM is remaining to store memtx tuples and indexes for an instance. If Tarantool hit the limits, the instance will become unavailable for write operations, so this alert may help you see when it's time to increase your `memtx_memory` limit or to add a new storage to a vshard cluster.

```
- alert: LowMemtxArenaRemainingWarning
  expr: (tnt_slab_quota_used_ratio >= 80) and (tnt_slab_arena_used_ratio >= 80)
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') low arena memory␣
↪remaining"
    description: "Low arena memory (tuples and indexes) remaining for '{{ $labels.alias }
↪}' instance of job '{{ $labels.job }}'.
      Consider increasing memtx_memory or number of storages in case of sharded data."

- alert: LowMemtxArenaRemaining
  expr: (tnt_slab_quota_used_ratio >= 90) and (tnt_slab_arena_used_ratio >= 90)
  for: 1m
  labels:
    severity: page
  annotations:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') low arena memory␣
↪remaining"
    description: "Low arena memory (tuples and indexes) remaining for '{{ $labels.alias }
↪}' instance of job '{{ $labels.job }}'.
      You are likely to hit limit soon.
      It is strongly recommended to increase memtx_memory or number of storages in case␣
↪of sharded data."

- alert: LowMemtxItemsRemainingWarning
  expr: (tnt_slab_quota_used_ratio >= 80) and (tnt_slab_items_used_ratio >= 80)
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') low items memory␣
↪remaining"
    description: "Low items memory (tuples) remaining for '{{ $labels.alias }}' instance␣
↪of job '{{ $labels.job }}'.
      Consider increasing memtx_memory or number of storages in case of sharded data."

- alert: LowMemtxItemsRemaining
  expr: (tnt_slab_quota_used_ratio >= 90) and (tnt_slab_items_used_ratio >= 90)
  for: 1m
  labels:
    severity: page
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') low items memory␣
↪remaining"
    description: "Low items memory (tuples) remaining for '{{ $labels.alias }}' instance␣
↪of job '{{ $labels.job }}'.
      You are likely to hit limit soon.
      It is strongly recommended to increase memtx_memory or number of storages in case␣
↪of sharded data."
```

### Vinyl engine status

You can monitor *vinyl regulator* performance to track possible scheduler or disk issues.

```
- alert: LowVinylRegulatorRateLimit
  expr: tnt_vinyl_regulator_rate_limit < 100000
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') have low vinyl␣
↪regulator rate limit"
    description: "Instance '{{ $labels.alias }}' of job '{{ $labels.job }}' have low␣
↪vinyl engine regulator rate limit.
      This indicates issues with the disk or the scheduler."
```

*Vinyl transactions* errors are likely to lead to user requests errors.

```
- alert: HighVinylTxConflictRate
  expr: rate(tnt_vinyl_tx_conflict[5m]) / rate(tnt_vinyl_tx_commit[5m]) > 0.05
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') have high vinyl tx␣
↪conflict rate"
    description: "Instance '{{ $labels.alias }}' of job '{{ $labels.job }}' have
      high vinyl transactions conflict rate. It indicates that vinyl is not healthy."
```

*Vinyl scheduler* failed tasks are a good signal of disk issues and may be the reason of increasing RAM consumption.

```
- alert: HighVinylSchedulerFailedTasksRate
  expr: rate(tnt_vinyl_scheduler_tasks{status="failed"}[5m]) > 0.1
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') have high vinyl␣
↪scheduler failed tasks rate"
    description: "Instance '{{ $labels.alias }}' of job '{{ $labels.job }}' have
      high vinyl scheduler failed tasks rate."
```

### Replication state

If `tnt_replication_status` is equal to 0, instance *replication* status is not equal to `"follows"`: replication is either not ready yet or has been stopped due to some reason.

```
- alert: ReplicationNotRunning
  expr: tnt_replication_status == 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') {{ $labels.stream }}␣
↪(id {{ $labels.id }})
      replication is not running"
    description: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') {{ $labels.stream␣
↪}} (id {{ $labels.id }})
      replication is not running. Check Cartridge UI for details."
```

Even if async replication is `"follows"`, it could be considered malfunctioning if the lag is too high. It also may affect Tarantool garbage collector work, see *box.info.gc()*.

```
- alert: HighReplicationLag
  expr: tnt_replication_lag > 1
  for: 1m
  labels:
    severity: warning
  annotations:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') have high replication␣
↪lag (id {{ $labels.id }})"
    description: "Instance '{{ $labels.alias }}' of job '{{ $labels.job }}' have high␣
↪replication lag
      (id {{ $labels.id }}), check up your network and cluster state."
```

### Event loop

High *fiber* event loop time leads to bad application performance, timeouts and various warnings. The reason could be a high quantity of working fibers or fibers that spend too much time without any yields or sleeps.

```
- alert: HighEVLoopTime
  expr: tnt_ev_loop_time > 0.1
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') event loop has high␣
↪cycle duration"
    description: "Instance '{{ $labels.alias }}' of job '{{ $labels.job }}' event loop␣
↪has high cycle duration.
      Some high loaded fiber has too little yields. It may be the reason of 'Too long␣
↪WAL write' warnings."
```

### Cartridge issues

Cartridge issues and warnings aggregate both single instance or replicaset issues (like memory or replication issues we've discussed in another paragraphs) and Cartridge cluster malfunctions (for example, clusteride config issues).

```
- alert: CartridgeWarningIssues
  expr: tnt_cartridge_issues{level="warning"} > 0
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') has 'warning'-level␣
↪Cartridge issues"
    description: "Instance '{{ $labels.alias }}' of job '{{ $labels.job }}' has 'warning
↪'-level Cartridge issues.
      Possible reasons: high replication lag, replication long idle,
      failover or switchover issues, clock issues, memory fragmentation,
      configuration issues, alien members."

- alert: CartridgeCriticalIssues
  expr: tnt_cartridge_issues{level="critical"} > 0
  for: 1m
  labels:
    severity: page
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') has 'critical'-level␣
↪Cartridge issues"
    description: "Instance '{{ $labels.alias }}' of job '{{ $labels.job }}' has 'critical
↪'-level Cartridge issues.
      Possible reasons: replication process critical fail,
      running out of available memory."
```

### HTTP server statistics

*metrics* allows to monitor tarantool/http handles, see *«Collecting HTTP request latency statistics»*. Here we use a `summary` collector with a default name and 0.99 quantile computation.

Too many responses with error codes usually is a sign of API issues or application malfunction.

```
- alert: HighInstanceHTTPClientErrorRate
  expr: sum by (job, instance, method, path, alias) (rate(http_server_request_latency_
↪count{ job="tarantool", status=~"^4\\d{2}$" }[5m])) > 10
  for: 1m
  labels:
    severity: page
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') high rate of client␣
↪error responses"
    description: "Too many {{ $labels.method }} requests to {{ $labels.path }} path
      on '{{ $labels.alias }}' instance of job '{{ $labels.job }}' get client error␣
↪(4xx) responses."

- alert: HighHTTPClientErrorRate
  expr: sum by (job, method, path) (rate(http_server_request_latency_count{ job=
↪"tarantool", status=~"^4\\d{2}$" }[5m])) > 20
  for: 1m
  labels:
    severity: page
  annotations:
    summary: "Job '{{ $labels.job }}' high rate of client error responses"
    description: "Too many {{ $labels.method }} requests to {{ $labels.path }} path
      on instances of job '{{ $labels.job }}' get client error (4xx) responses."

- alert: HighHTTPServerErrorRate
  expr: sum by (job, instance, method, path, alias) (rate(http_server_request_latency_
↪count{ job="tarantool", status=~"^5\\d{2}$" }[5m])) > 0
  for: 1m
  labels:
    severity: page
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') server error responses
↪"
    description: "Some {{ $labels.method }} requests to {{ $labels.path }} path
      on '{{ $labels.alias }}' instance of job '{{ $labels.job }}' get server error␣
↪(5xx) responses."
```

Responding with high latency is a synonym of insufficient performance. It may be a sign of application malfunction. Or maybe you need to add more routers to your cluster.

```
- alert: HighHTTPLatency
  expr: http_server_request_latency{ job="tarantool", quantile="0.99" } > 0.1
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') high HTTP latency"
    description: "Some {{ $labels.method }} requests to {{ $labels.path }} path with {{
↪$labels.status }} response status
      on '{{ $labels.alias }}' instance of job '{{ $labels.job }}' are processed too␣
↪long."
```

Having too little requests when you expect them may detect balancer, external client or network malfunction.

```
- alert: LowRouterHTTPRequestRate
  expr: sum by (job, instance, alias) (rate(http_server_request_latency_count{ job=
↪"tarantool", alias=~"^.*router.*$" }[5m])) < 10
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "Router '{{ $labels.alias }}' ('{{ $labels.job }}') low activity"
    description: "Router '{{ $labels.alias }}' instance of job '{{ $labels.job }}' gets␣
↪too little requests.
      Please, check up your balancer middleware."
```

### CRUD module statistics

If your application uses CRUD module requests, monitoring module statistics may track internal errors caused by invalid process of input and internal parameters.

```
- alert: HighCRUDErrorRate
  expr: rate(tnt_crud_stats_count{ job="tarantool", status="error" }[5m]) > 0.1
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') too many CRUD {{
↪$labels.operation }} errors."
    description: "Too many {{ $labels.operation }} CRUD requests for '{{ $labels.name }}
↪' space on
      '{{ $labels.alias }}' instance of job '{{ $labels.job }}' get module error␣
↪responses."
```

Statistics could also monitor requests performance. Too high request latency will lead to high latency of client responses. It may be caused by network or disk issues. Read requests with bad (with respect to space indexes and sharding schema) conditions may lead to full-scans or map reduces and also could be the reason of high latency.

```
- alert: HighCRUDLatency
  expr: tnt_crud_stats{ job="tarantool", quantile="0.99" } > 0.1
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') too high CRUD {{
↪$labels.operation }} latency."
    description: "Some {{ $labels.operation }} {{ $labels.status }} CRUD requests for '{
↪{ $labels.name }}' space on
      '{{ $labels.alias }}' instance of job '{{ $labels.job }}' are processed too long."
```

You also can directly monitor map reduces and scan rate.

```
- alert: HighCRUDMapReduceRate
  expr: rate(tnt_crud_map_reduces{ job="tarantool" }[5m]) > 0.1
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Instance '{{ $labels.alias }}' ('{{ $labels.job }}') too many CRUD {{
↪$labels.operation }} map reduces."
    description: "There are too many {{ $labels.operation }} CRUD map reduce requests␣
↪for '{{ $labels.name }}' space on
      '{{ $labels.alias }}' instance of job '{{ $labels.job }}'.
     Check your request conditions or consider changing sharding schema."
```

### Server-side monitoring

If there are no Tarantool metrics, you may miss critical conditions. Prometheus provide `up` metric to monitor the health of its targets.

```
- alert: InstanceDown
  expr: up == 0
  for: 1m
  labels:
    severity: page
  annotations:
    summary: "Instance '{{ $labels.instance }}' ('{{ $labels.job }}') down"
    description: "'{{ $labels.instance }}' of job '{{ $labels.job }}' has been down for␣
↪more than a minute."
```

Do not forget to monitor your server's CPU, disk and RAM from server side with your favorite tools. For example, on some high CPU consumption cases Tarantool instance may stop to send metrics, so you can track such breakdowns only from the outside.

ГЛАВА 7

# Коннекторы

Коннекторы — это API, позволяющие использовать Tarantool с различными языками программирования.

Connectors can be divided into two groups – those maintained by the Tarantool team and those supported by the community. The Tarantool team maintains the *high-level C API*, the *Go* and *Java* connectors, and a synchronous *Python* connector. All other connectors are *community-supported*, which means that support for new Tarantool features may be delayed. Besides, the Tarantool support team cannot prioritize issues that arise while working through these connectors.

This chapter documents APIs for various programming languages:

## 7.1 C

В этом разделе даны два примера использования высокоуровневого API для Tarantool и языка C.

### 7.1.1 Пример 1

Далее приводится пример полноценной программы на языке C, которая осуществляет вставку кортежа `[99999,'B']` в спейс `examples` с помощью высокоуровневого API для языка C.

```
#include <stdio.h>
#include <stdlib.h>

#include <tarantool/tarantool.h>
#include <tarantool/tnt_net.h>
#include <tarantool/tnt_opt.h>

void main() {
   struct tnt_stream *tnt = tnt_net(NULL);          /* См. ниже = НАСТРОЙКА */
   tnt_set(tnt, TNT_OPT_URI, "localhost:3301");
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    if (tnt_connect(tnt) < 0) {                         /* См. ниже = СОЕДИНЕНИЕ */
        printf("Connection refused\n");
        exit(-1);
    }
    struct tnt_stream *tuple = tnt_object(NULL);        /* См. ниже = СОЗДАНИЕ ЗАПРОСА */
    tnt_object_format(tuple, "[%d%s]", 99999, "B");
    tnt_insert(tnt, 999, tuple);                        /* См. ниже = ОТПРАВКА ЗАПРОСА */
    tnt_flush(tnt);
    struct tnt_reply reply;  tnt_reply_init(&reply); /* См. ниже = ПОЛУЧЕНИЕ ОТВЕТА */
    tnt->read_reply(tnt, &reply);
    if (reply.code != 0) {
        printf("Insert failed %lu.\n", reply.code);
    }
    tnt_close(tnt);                                     /* См. ниже = ЗАВЕРШЕНИЕ */
    tnt_stream_free(tuple);
    tnt_stream_free(tnt);
}
```

Скопируйте исходный код программы в файл с именем `example.c` и установите коннектор `tarantool-c`. Вот один из способов установки `tarantool-c` (под Ubuntu):

```
$ git clone git://github.com/tarantool/tarantool-c.git ~/tarantool-c
$ cd ~/tarantool-c
$ git submodule init
$ git submodule update
$ cmake .
$ make
$ make install
```

To compile and link the program, run:

```
$ # иногда это необходимо:
$ export LD_LIBRARY_PATH=/usr/local/lib
$ gcc -o example example.c -ltarantool
```

Before trying to run, check that a server instance is listening at `localhost:3301` and that the space `examples` exists, as *described earlier*. To run the program, say `./example`. The program will connect to the Tarantool instance, and will send the request. If Tarantool is not running on localhost with listen address = 3301, the program will print “Connection refused”. If the insert fails, the program will print «Insert failed» and an error number (see all error codes in the source file /src/box/errcode.h).

Далее следуют примечания, на которые мы ссылались в комментариях к исходному коду тестовой программы.

### SETUP

The setup begins by creating a stream.

```
struct tnt_stream *tnt = tnt_net(NULL);
tnt_set(tnt, TNT_OPT_URI, "localhost:3301");
```

В нашей программе поток назван `tnt`. Перед установкой соединения с потоком `tnt` нужно задать ряд опций. Самая важная из них – TNT_OPT_URI. Для этой опции указан *URI* `localhost:3301`, т.е. адрес, по которому должно быть настроено прослушивание на стороне экземпляра Tarantool.

Описание функции:

```
struct tnt_stream *tnt_net(struct tnt_stream *s)
int tnt_set(struct tnt_stream *s, int option, variant option-value)
```

### CONNECT

Now that the stream named `tnt` exists and is associated with a URI, this example program can connect to a server instance.

```
if (tnt_connect(tnt) < 0)
   { printf("Connection refused\n"); exit(-1); }
```

Описание функции:

```
int tnt_connect(struct tnt_stream *s)
```

Попытка соединения может и не удаться по разным причинам, например если Tarantool-сервер не запущен или в URI-строке указан неверный *пароль*. В случае неудачи функция вернет -1.

### MAKE REQUEST

Most requests require passing a structured value, such as the contents of a tuple.

```
struct tnt_stream *tuple = tnt_object(NULL);
tnt_object_format(tuple, "[%d%s]", 99999, "B");
```

В данной программе мы используем запрос *INSERT*, а кортеж содержит целое число и строку. Это простой набор значений без каких-либо вложенных структур или массивов. И передаваемые значения мы можем указать самым простым образом – аналогично тому, как это сделано в стандартной C-функции `printf()`: `%d` для обозначения целого числа, `%s` для обозначения строки, затем числовое значение, затем указатель на строковое значение.

Описание функции:

```
ssize_t tnt_object_format(struct tnt_stream *s, const char *fmt, ...)
```

### SEND REQUEST

The database-manipulation requests are analogous to the requests in the box library.

```
tnt_insert(tnt, 999, tuple);
tnt_flush(tnt);
```

В данной программе мы делаем INSERT-запрос. В этом запросе мы передаем поток `tnt`, который ранее использовали для установки соединения, и поток `tuple`, который также ранее настроили с помощью функции `tnt_object_format()`.

Описание функции:

```
ssize_t tnt_insert(struct tnt_stream *s, uint32_t space, struct tnt_stream *tuple)
ssize_t tnt_replace(struct tnt_stream *s, uint32_t space, struct tnt_stream *tuple)
ssize_t tnt_select(struct tnt_stream *s, uint32_t space, uint32_t index,
                   uint32_t limit, uint32_t offset, uint8_t iterator,
                   struct tnt_stream *key)
ssize_t tnt_update(struct tnt_stream *s, uint32_t space, uint32_t index,
                   struct tnt_stream *key, struct tnt_stream *ops)
```

### GET REPLY

For most requests, the client will receive a reply containing some indication whether the result was successful, and a set of tuples.

```
struct tnt_reply reply;  tnt_reply_init(&reply);
tnt->read_reply(tnt, &reply);
if (reply.code != 0)
   { printf("Insert failed %lu.\n", reply.code); }
```

Данная программа проверяет, был ли запрос выполнен успешно, но никак не интерпретирует оставшуюся часть ответа.

Описание функции:

```
struct tnt_reply *tnt_reply_init(struct tnt_reply *r)
tnt->read_reply(struct tnt_stream *s, struct tnt_reply *r)
void tnt_reply_free(struct tnt_reply *r)
```

### TEARDOWN

When a session ends, the connection that was made with `tnt_connect()` should be closed, and the objects that were made in the setup should be destroyed.

```
tnt_close(tnt);
tnt_stream_free(tuple);
tnt_stream_free(tnt);
```

Описание функции:

```
void tnt_close(struct tnt_stream *s)
void tnt_stream_free(struct tnt_stream *s)
```

### 7.1.2 Пример 2

Here is a complete C program that selects, using index key `[99999]`, from space `examples` via the high-level C API. To display the results, the program uses functions in the MsgPuck library which allow decoding of MessagePack arrays.

```
#include <stdio.h>
#include <stdlib.h>
#include <tarantool/tarantool.h>
#include <tarantool/tnt_net.h>
#include <tarantool/tnt_opt.h>

#define MP_SOURCE 1
#include <msgpuck.h>

void main() {
    struct tnt_stream *tnt = tnt_net(NULL);
    tnt_set(tnt, TNT_OPT_URI, "localhost:3301");
    if (tnt_connect(tnt) < 0) {
        printf("Connection refused\n");
        exit(1);
    }
    struct tnt_stream *tuple = tnt_object(NULL);
    tnt_object_format(tuple, "[%d]", 99999); /* кортеж tuple = ключ для поиска */
    tnt_select(tnt, 999, 0, (2^32) - 1, 0, 0, tuple);
    tnt_flush(tnt);
    struct tnt_reply reply; tnt_reply_init(&reply);
    tnt->read_reply(tnt, &reply);
    if (reply.code != 0) {
        printf("Select failed.\n");
        exit(1);
    }
    char field_type;
    field_type = mp_typeof(*reply.data);
    if (field_type != MP_ARRAY) {
        printf("no tuple array\n");
        exit(1);
    }
    long unsigned int row_count;
    uint32_t tuple_count = mp_decode_array(&reply.data);
    printf("tuple count=%u\n", tuple_count);
    unsigned int i, j;
    for (i = 0; i < tuple_count; ++i) {
        field_type = mp_typeof(*reply.data);
        if (field_type != MP_ARRAY) {
            printf("no field array\n");
            exit(1);
        }
        uint32_t field_count = mp_decode_array(&reply.data);
        printf("  field count=%u\n", field_count);
        for (j = 0; j < field_count; ++j) {
            field_type = mp_typeof(*reply.data);
            if (field_type == MP_UINT) {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                uint64_t num_value = mp_decode_uint(&reply.data);
                printf("    value=%lu.\n", num_value);
            } else if (field_type == MP_STR) {
                const char *str_value;
                uint32_t str_value_length;
                str_value = mp_decode_str(&reply.data, &str_value_length);
                printf("    value=%.*s.\n", str_value_length, str_value);
            } else {
                printf("wrong field type\n");
                exit(1);
            }
        }
    }
    tnt_close(tnt);
    tnt_stream_free(tuple);
    tnt_stream_free(tnt);
}
```

Аналогично первому примеру, сохраните исходный код программы в файле с именем `example2.c`.

Чтобы скомпилировать и слинковать тестовую программу, выполните следующую команду:

```
$ gcc -o example2 example2.c -ltarantool
```

Для запуска программы выполните команду `./example2`.

The two example programs only show a few requests and do not show all that's necessary for good practice. See more in the tarantool-c documentation at GitHub.

## 7.2 Go

The following connectors are available:

- Tarantool-supported go-tarantool
- Community-supported viciious/go-tarantool
- Community-supported FZambia/tarantool.

### 7.2.1 Feature comparison

Last update: January 2023

| | tarantool/go-tarantool | viciious/go-tar |
|---|---|---|
| License | BSD 2-Clause | MIT |
| Last update | 2023 | 2022 |
| Documentation | README with examples and up-to-date GoDoc | README wit |
| Testing / CI / CD | GitHub Actions | Travis CI |
| GitHub Stars | 147 | 45 |
| Static analysis | golangci-lint, luacheck | golint |
| Packaging | go get | go get |

| | tarantool/go-tarantool | viciious/go-tar |
|---|---|---|
| Code coverage | Yes | No |
| msgpack driver | vmihailenco/msgpack/v2 or vmihailenco/msgpack/v5 | tinylib/msgp |
| Async work | Yes | Yes |
| Schema reload | Yes (manual pull) | Yes (manual p |
| Space / index names | Yes | Yes |
| Tuples as structures | Yes (structure and marshall functions must be predefined in Go code) | No |
| Access tuple fields by names | Only if marshalled to structure | No |
| SQL support | Yes | No (#18, clos |
| Interactive transactions | Yes | No |
| Varbinary support | Yes (with in-built language tools) | Yes (with in-b |
| UUID support | Yes | No |
| Decimal support | Yes | No |
| EXT_ERROR support | Yes | No |
| Datetime support | Yes | No |
| box.session.push() responses | Yes | No (#21) |
| Session settings | Yes | No |
| Graceful shutdown | Yes | No |
| IPROTO_ID (feature discovering) | Yes | No |
| tarantool/crud support | No | No |
| Connection pool | Yes (round-robin failover, no balancing) | No |
| Transparent reconnecting | Yes (see comments in #129) | No (handle re |
| Transparent request retrying | No | No |
| Watchers | Yes | No |
| Pagination | Yes | No |
| Language features | context | context |
| Miscellaneous | Supports tarantool/queue API | Can mimic a |

## 7.3 Java

Доступны два Java-коннектора:

- cartridge-java supports both single Tarantool nodes and clusters, as well as applications built using the Cartridge framework and its modules. The Tarantool team actively updates this module with the newest Tarantool features.
- tarantool-java предназначен для ранних версий Tarantool (1.6 и выше) и предоставляет интерфейс JDBC для одиночных узлов Tarantool. Этот модуль *в настоящее время не поддерживается* и не работает ни с новейшими функциональными возможностями Tarantool 2.x, ни с кластерами Tarantool.

Для работы с библиотеками и фреймворками Java в Tarantool существуют следующие модули:

- TestContainers Tarantool module adds support for the popular TestContainers framework used for integration testing of Java applications.
- Модуль Spring Data в Tarantool добавляет поддержку фреймворка Spring.

  Чтобы узнать, как использовать этот модуль в реальных приложениях, ознакомьтесь с проектом Spring Pet Clinic.

## 7.4 Python

tarantool-python is the official Python connector for Tarantool. It is not supplied as part of the Tarantool repository and must be installed separately (see below for details).

Далее приводится пример полноценной программы на языке Python, которая осуществляет вставку `[99999,'Value','Value']` в спейс `examples` с помощью высокоуровневого API для языка Python.

```
#!/usr/bin/python
from tarantool import Connection

c = Connection("127.0.0.1", 3301)
result = c.insert("examples",(99999,'Value', 'Value'))
print result
```

Чтобы запустить тестовую программу, сохраните ее исходный код в файл с именем `example.py` и установите коннектор `tarantool-python`. Для установки коннектора воспользуйтесь либо командой `pip install tarantool>0.4` (для установки в директорию `/usr`; вам потребуются права уровня **root**), либо командой `pip install tarantool>0.4 --user` (для установки в директорию `~`, т.е. в используемую по умолчанию директорию текущего пользователя).

Before trying to run, check that the server instance is *listening* at `localhost:3301` and that the space `examples` exists, as *described earlier*. To run the program, say `python example.py`. The program will connect to the Tarantool server, will send the *INSERT* request, and will not throw any exception if all went well. If the tuple already exists, the program will throw `tarantool.error.DatabaseError: (3, "Duplicate key exists in unique index 'primary' in space 'examples'")`.

The example program only shows one request and does not show all that's necessary for good practice. For that, please see tarantool-python project at GitHub.

Кроме того, сообщество разработчиков поддерживает другие Python-коннекторы:

- asynctnt with asyncio support
- aiotarantool also with asyncio support, **no active maintenance**
- gtarantool with gevent support, **no active maintenance**

The table below contains a feature comparison for asynctnt and tarantool-python. aiotarantool and gtarantool are absent there because they are quite outdated and unmaintained.

### 7.4.1 Feature comparison

Last update: September 2023

| Parameter | igorcoding/asynctnt | tarantool/ |
|---|---|---|
| License | Apache License 2.0 | BSD-2 |
| Is maintained | Yes | Yes |
| Known Issues | None | None |
| Documentation | Yes (github.io) | Yes (readt |
| Testing / CI / CD | GitHub Actions | GitHub A |
| GitHub Stars | 73 | 92 |
| Static Analysis | Yes (Flake8) | Yes (Flak |
| Packaging | pip | pip, deb, |
| Code coverage | Yes | Yes |

Таблица 2 – продолжение с предыдущей страницы

| Parameter | igorcoding/asynctnt | tarantool/ |
|---|---|---|
| Support asynchronous mode | Yes, asyncio | No |
| Batching support | No | Yes (with |
| Schema reload | Yes (automatically, see auto_refetch_schema) | Yes (auto |
| Space / index names | Yes | Yes |
| Access tuple fields by names | Yes | No |
| *SQL support* | Yes | Yes |
| *Interactive transactions* | Yes | No (issue |
| *Varbinary support* | Yes (in `MP_BIN` fields) | Yes |
| *Decimal support* | Yes | Yes |
| *UUID support* | Yes | Yes |
| *EXT_ERROR support* | Yes | Yes |
| *Datetime support* | Yes | Yes |
| *Interval support* | No (issue #30) | Yes |
| *box.session.push() responses* | Yes | Yes |
| *Session settings* | No | No |
| Graceful shutdown | No | No |
| IPROTO_ID (feature discovery) | Yes | Yes |
| CRUD support | No | Yes |
| Transparent request retrying | No | No |
| Transparent reconnecting | Autoreconnect | Yes (recon |
| Connection pool | No | Yes (with |
| Support of PEP 249 – Python Database API Specification v2.0 | No | Yes |
| *Encrypted connection (Enterprise Edition)* | No (issue #22) | Yes |

## 7.5 Community-supported connectors

This section provides information on several community-supported connectors. Note that they may have limited support for new Tarantool features.

### 7.5.1 C#

The most commonly used C# driver is progaudi.tarantool, previously named `tarantool-csharp`. It is not supplied as part of the Tarantool repository; it must be installed separately. The makers recommend cross-platform installation using Nuget.

Чтобы придерживаться метода оформления других инструкций в данной главе, дадим описание способа установки драйвера напрямую на 16.04.

1. Install .net core from Microsoft. Follow .net core installation instructions.

---

**Примечание:**

- Mono не сработает, как не сработает и .Net от xbuild. Только .NET Core поддерживается на Linux и Mac.
- Read the Microsoft End User License Agreement first, because it is not an ordinary open-source agreement and there will be a message during installation saying «This software may collect information about you and your use of the software, and send that to Microsoft.» Still you can set environment variables to opt out from telemetry.

---

2. Создайте новый консольный проект.

```
$ cd ~
$ mkdir progaudi.tarantool.test
$ cd progaudi.tarantool.test
$ dotnet new console
```

3. Добавьте ссылку на `progaudi.tarantool`.

```
$ dotnet add package progaudi.tarantool
```

4. Измените код в `Program.cs`.

```
$ cat <<EOT > Program.cs
using System;
using System.Threading.Tasks;
using ProGaudi.Tarantool.Client;

public class HelloWorld
{
  static public void Main ()
  {
    Test().GetAwaiter().GetResult();
  }
  static async Task Test()
  {
    var box = await Box.Connect("127.0.0.1:3301");
    var schema = box.GetSchema();
    var space = await schema.GetSpace("examples");
    await space.Insert((99999, "BB"));
  }
}
EOT
```

5. Соберите и запустите приложение.

   Перед запуском проверьте, что у экземпляра задан порт для прослушивания на``localhost:3301``, и в базе создан спейс `examples`, как *описано выше*.

```
$ dotnet restore
$ dotnet run
```

   Программа:

   - установит соединение, используя определение спейса для этой цели,
   - open a socket connection with the Tarantool server at `localhost:3301`,
   - отправит INSERT-запрос, а затем – если всё хорошо – закончит работу без каких-либо сообщений.

   Если Tarantool не запущен на `localhost` на прослушивание по порту 3301, или у пользователя „guest“ нет прав на соединение, или запрос вставки по какой-либо причине не сработает, то программа выдаст сообщение об ошибке и другую информацию (трассировку стека и т.д.).

The example program only shows one request and does not show all that's necessary for good practice. For that, please see the progaudi.tarantool driver repository.

### 7.5.2 C++ connector API

The official C++ connector for Tarantool is located in the tanartool/tntcxx repository.

It is not supplied as part of the Tarantool repository and requires additional actions for usage. The connector itself is a header-only library and, as such, doesn't require installation and building. All you need is to clone the connector source code and embed it in your C++ project. See the C++ connector Getting started document for details and examples.

Below is the description of the connector public API.


- *Connector class*
  - *Public methods*
- *Connection class*
  - *Public types*
  - *Public methods*
  - *Nested classes and their methods*


#### Connector class

template<class `BUFFER`, class `NetProvider` = EpollNetProvider<*BUFFER*>>
class `Connector`

The `Connector` class is a template class that defines a connector client which can handle many connections to Tarantool instances asynchronously.

To instantiate a client, you should specify the buffer and the network provider implementations as template parameters. You can either implement your own buffer or network provider or use the default ones.

The default connector instantiation looks as follows:

```
using Buf_t = tnt::Buffer<16 * 1024>;
using Net_t = EpollNetProvider<Buf_t >;
Connector<Buf_t, Net_t> client;
```

#### Public methods

- *connect()*
- *wait()*
- *waitAll()*
- *waitAny()*
- *close()*

int `connect`(*Connection*<BUFFER, NetProvider> &conn, const std::string_view &addr, unsigned port,
size_t timeout = DEFAULT_CONNECT_TIMEOUT)

Connects to a Tarantool instance that is listening on `addr:port`. On successful connection, the method returns `0`. If the host doesn't reply within the timeout period or another error occurs, it returns `-1`. Then, *Connection.getError()* gives the error message.

**Параметры**

- `conn` – object of the *Connection* class.
- `addr` – address of the host where a Tarantool instance is running.
- `port` – port that a Tarantool instance is listening on.
- `timeout` – connection timeout, seconds. Optional. Defaults to 2.

**Результат** `0` on success, or `-1` otherwise.

**Rtype** int

**Possible errors:**

- connection timeout
- refused to connect (due to incorrect address or/and port)
- system errors: a socket can't be created; failure of any of the system calls (`fcntl`, `select`, `send`, `receive`).

**Example:**

```
using Buf_t = tnt::Buffer<16 * 1024>;
using Net_t = EpollNetProvider<Buf_t >;

Connector<Buf_t, Net_t> client;
Connection<Buf_t, Net_t> conn(client);

int rc = client.connect(conn, "127.0.0.1", 3301);
```

int `wait`(*Connection*<BUFFER, NetProvider> &conn, *rid_t* future, int timeout = 0)

The main method responsible for sending a request and checking the response readiness.

You should prepare a request beforehand by using the necessary method of the *Connection* class, such as *ping()* and so on, which encodes the request in the MessagePack format and saves it in the output connection buffer.

`wait()` sends the request and is polling the `future` for the response readiness. Once the response is ready, `wait()` returns `0`. If at `timeout` the response isn't ready or another error occurs, it returns `-1`. Then, *Connection.getError()* gives the error message. `timeout = 0` means the method is polling the `future` until the response is ready.

**Параметры**

- `conn` – object of the *Connection* class.
- `future` – request ID returned by a request method of the *Connection* class, such as, *ping()* and so on.
- `timeout` – waiting timeout, milliseconds. Optional. Defaults to `0`.

**Результат** `0` on receiving a response, or `-1` otherwise.

**Rtype** int

**Possible errors:**

- timeout exceeded
- other possible errors depend on a network provider used. If the `EpollNetProvider` is used, failing of the `poll`, `read`, and `write` system calls leads to system errors, such as, `EBADF`, `ENOTSOCK`, `EFAULT`, `EINVAL`, `EPIPE`, and `ENOTCONN` (`EWOULDBLOCK` and `EAGAIN` don't occur in this case).

**Example:**

```
client.wait(conn, ping, WAIT_TIMEOUT)
```

void `waitAll`(*Connection*<BUFFER, NetProvider> &conn, *rid_t* *futures, size_t future_count, int timeout = 0)

Similar to *wait()*, the method sends the requests prepared and checks the response readiness, but can send several different requests stored in the `futures` array. Exceeding the timeout leads to an error; *Connection.getError()* gives the error message. `timeout = 0` means the method is polling the `futures` until all the responses are ready.

**Параметры**

- `conn` – object of the *Connection* class.
- `futures` – array with the request IDs returned by request methods of the *Connection* class, such as, *ping()* and so on.
- `future_count` – size of the `futures` array.
- `timeout` – waiting timeout, milliseconds. Optional. Defaults to `0`.

**Результат** none

**Rtype** none

**Possible errors:**

- timeout exceeded
- other possible errors depend on a network provider used. If the `EpollNetProvider` is used, failing of the `poll`, `read`, and `write` system calls leads to system errors, such as, `EBADF`, `ENOTSOCK`, `EFAULT`, `EINVAL`, `EPIPE`, and `ENOTCONN` (`EWOULDBLOCK` and `EAGAIN` don't occur in this case).

**Example:**

```
rid_t futures[2];
futures[0] = replace;
futures[1] = select;

client.waitAll(conn, (rid_t *) &futures, 2);
```

*Connection*<BUFFER, NetProvider> *`waitAny`(int timeout = 0)

Sends all requests that are prepared at the moment and is waiting for any first response to be ready. Upon the response readiness, `waitAny()` returns the corresponding connection object. If at `timeout` no response is ready or another error occurs, it returns `nullptr`. Then, *Connection.getError()* gives the error message. `timeout = 0` means no time limitation while waiting for the response readiness.

**Параметры** `timeout` – waiting timeout, milliseconds. Optional. Defaults to `0`.

**Результат** object of the *Connection* class on success, or `nullptr` on error.

**Rtype** Connection<BUFFER, NetProvider>*

**Possible errors:**

- timeout exceeded
- other possible errors depend on a network provider used. If the `EpollNetProvider` is used, failing of the `poll`, `read`, and `write` system calls leads to system errors, such as, `EBADF`, `ENOTSOCK`, `EFAULT`, `EINVAL`, `EPIPE`, and `ENOTCONN` (`EWOULDBLOCK` and `EAGAIN` don't occur in this case).

**Example:**

```
rid_t f1 = conn.ping();
rid_t f2 = another_conn.ping();

Connection<Buf_t, Net_t> *first = client.waitAny(WAIT_TIMEOUT);
if (first == &conn) {
    assert(conn.futureIsReady(f1));
} else {
    assert(another_conn.futureIsReady(f2));
}
```

void `close`(*Connection*<BUFFER, NetProvider> &conn)

Closes the connection established earlier by the *connect()* method.

**Параметры** `conn` – connection object of the *Connection* class.

**Результат** none

**Rtype** none

**Possible errors:** none.

**Example:**

```
client.close(conn);
```

### Connection class

template<class `BUFFER`, class `NetProvider`>
class `Connection`

The `Connection` class is a template class that defines a connection objects which is required to interact with a Tarantool instance. Each connection object is bound to a single socket.

Similar to a *connector client*, a connection object also takes the buffer and the network provider as template parameters, and they must be the same as ones of the client. For example:

```
//Instantiating a connector client
using Buf_t = tnt::Buffer<16 * 1024>;
using Net_t = EpollNetProvider<Buf_t >;
Connector<Buf_t, Net_t> client;

//Instantiating connection objects
Connection<Buf_t, Net_t> conn01(client);
Connection<Buf_t, Net_t> conn02(client);
```

The `Connection` class has two nested classes, namely, *Space* and *Index* that implement the data-manipulation methods like `select()`, `replace()`, and so on.


- *Public types*
- *Public methods*
- *Nested classes and their methods*

### Public types

typedef size_t `rid_t`
: The alias of the built-in `size_t` type. `rid_t` is used for entities that return or contain a request ID.

### Public methods

- *call()*
- *futureIsReady()*
- *getResponse()*
- *getError()*
- *reset()*
- *ping()*

template<class `T`>
*rid_t* `call`(const std::string &func, const *T* &args)
: Executes a call of a remote stored-procedure similar to *conn:call()*. The method returns the request ID that is used to get the response by *getResponse()*.

  **Параметры**

  - `func` – a remote stored-procedure name.
  - `args` – procedure's arguments.

  **Результат** a request ID

  **Rtype** rid_t

  **Possible errors:** none.

  **Example:**

  The following function is defined on the Tarantool instance you are connected to:

```
box.execute("DROP TABLE IF EXISTS t;")
box.execute("CREATE TABLE t(id INT PRIMARY KEY, a TEXT, b DOUBLE);")

function remote_replace(arg1, arg2, arg3)
    return box.space.T:replace({arg1, arg2, arg3})
end
```

  The function call can look as follows:

```
rid_t f1 = conn.call("remote_replace", std::make_tuple(5, "some_sring", 5.55));
```

bool `futureIsReady`(*rid_t* future)
: Checks availability of a request ID (`future`) returned by any of the request methods, such as, *ping()* and so on.

  `futureIsReady()` returns `true` if the `future` is available or `false` otherwise.

  **Параметры** `future` – a request ID.

  **Результат** `true` or `false`

  **Rtype** bool

**Possible errors:** none.

**Example:**

```
rid_t ping = conn.ping();
conn.futureIsReady(ping);
```

std::optional<Response<BUFFER>> `getResponse(`*rid_t* future)

The method takes a request ID (`future`) as an argument and returns an optional object containing a response. If the response is not ready, the method returns `std::nullopt`. Note that for each `future` the method can be called only once because it erases the request ID from the internal map as soon as the response is returned to a user.

A response consists of a header (`response.header`) and a body (`response.body`). Depending on success of the request execution on the server side, body may contain either runtime errors accessible by `response.body.error_stack` or data (tuples) accessible by `response.body.data`. Data is a vector of tuples. However, tuples are not decoded and come in the form of pointers to the start and the end of MessagePacks. For details on decoding the data received, refer to *«Decoding and reading the data»*.

**Параметры** `future` – a request ID

**Результат** a response object or `std::nullopt`

**Rtype** std::optional<Response<BUFFER>>

**Possible errors:** none.

**Example:**

```
rid_t ping = conn.ping();
std::optional<Response<Buf_t>> response = conn.getResponse(ping);
```

std::string &`getError()`

Returns an error message for the last error occured during the execution of methods of the *Connector* and *Connection* classes.

**Результат** an error message

**Rtype** std::string&

**Possible errors:** none.

**Example:**

```
int rc = client.connect(conn, address, port);

if (rc != 0) {
    assert(conn.status.is_failed);
    std::cerr << conn.getError() << std::endl;
    return -1;
}
```

void `reset()`

Resets a connection after errors, that is, cleans up the error message and the connection status.

**Результат** none

**Rtype** none

**Possible errors:** none.

**Example:**

```
if (client.wait(conn, ping, WAIT_TIMEOUT) != 0) {
    assert(conn.status.is_failed);
    std::cerr << conn.getError() << std::endl;
    conn.reset();
}
```

*rid_t* ping()
Prepares a request to ping a Tarantool instance.

The method encodes the request in the MessagePack format and queues it in the output connection buffer to be sent later by one of *Connector's* methods, namely, *wait()*, *waitAll()*, or *waitAny()*.

Returns the request ID that is used to get the response by the *getResponce()* method.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
rid_t ping = conn.ping();
```

### Nested classes and their methods

- *Space*
- *Index*

### Space class

class Space : *Connection*
Space is a nested class of the *Connection* class. It is a public wrapper to access the data-manipulation methods in the way similar to the Tarantool submodule *box.space*, like, space[space_id].select(), space[space_id].replace(), and so on.

All the Space class methods listed below work in the following way:

- A method encodes the corresponding request in the MessagePack format and queues it in the output connection buffer to be sent later by one of *Connector's* methods, namely, *wait()*, *waitAll()*, or *waitAny()*.
- A method returns the request ID. To get and read the actual data requested, first you need to get the response object by using the *getResponce()* method and then *decode* the data.

**Public methods**:

- *select()*
- *replace()*
- *insert()*
- *update()*
- *upsert()*
- *delete_()*

template<class T>
*rid_t* `select`(const *T* &key, uint32_t index_id = 0, uint32_t limit = UINT32_MAX, uint32_t offset = 0, IteratorType iterator = EQ)

Searches for a tuple or a set of tuples in the given space. The method works similar to *space_object:select()* and performs the search against the primary index (`index_id = 0`) by default. In other words, `space[space_id].select()` equals to `space[space_id].index[0].select()`.

**Параметры**

- `key` – value to be matched against the index key.
- `index_id` – index ID. Optional. Defaults to `0`.
- `limit` – maximum number of tuples to select. Optional. Defaults to `UINT32_MAX`.
- `offset` – number of tuples to skip. Optional. Defaults to `0`.
- `iterator` – the type of iterator. Optional. Defaults to `EQ`.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to space_object:select({key_value}, {limit = 1}) in Tarantool*/
uint32_t space_id = 512;
int key_value = 5;
uint32_t limit = 1;
auto i = conn.space[space_id];
rid_t select = i.select(std::make_tuple(key_value), index_id, limit, offset, iter);
```

template<class T>
*rid_t* `replace`(const *T* &tuple)

Inserts a tuple into the given space. If a tuple with the same primary key already exists, `replace()` replaces the existing tuple with a new one. The method works similar to *space_object:replace() / put()*.

**Параметры** `tuple` – a tuple to insert.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to space_object:replace(key_value, "111", 1.01) in Tarantool*/
uint32_t space_id = 512;
int key_value = 5;
std::tuple data = std::make_tuple(key_value, "111", 1.01);
rid_t replace = conn.space[space_id].replace(data);
```

template<class T>
*rid_t* `insert`(const *T* &tuple)

Inserts a tuple into the given space. The method works similar to *space_object:insert()*.

**Параметры** `tuple` – a tuple to insert.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to space_object:insert(key_value, "112", 2.22) in Tarantool*/
uint32_t space_id = 512;
int key_value = 6;
std::tuple data = std::make_tuple(key_value, "112", 2.22);
rid_t insert = conn.space[space_id].insert(data);
```

template<class K, class T>
*rid_t* update(const *K* &key, const *T* &tuple, uint32_t index_id = 0)

Updates a tuple in the given space. The method works similar to *space_object:update()* and searches for the tuple to update against the primary index (`index_id = 0`) by default. In other words, `space[space_id].update()` equals to `space[space_id].index[0].update()`.

The `tuple` parameter specifies an update operation, an identifier of the field to update, and a new field value. The set of available operations and the format of specifying an operation and a field identifier is the same as in Tarantool. Refer to the description of :doc:` </reference/reference_lua/box_space/update>` and example below for details.

**Параметры**

- `key` – value to be matched against the index key.
- `tuple` – parameters for the update operation, namely, `operator, field_identifier, value`.
- `index_id` – index ID. Optional. Defaults to `0`.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to space_object:update(key, {{'=', 1, 'update' }, {'+', 2, 12}}) in␣
↪Tarantool*/
uint32_t space_id = 512;
std::tuple key = std::make_tuple(5);
std::tuple op1 = std::make_tuple("=", 1, "update");
std::tuple op2 = std::make_tuple("+", 2, 12);
rid_t f1 = conn.space[space_id].update(key, std::make_tuple(op1, op2));
```

template<class T, class O>
*rid_t* upsert(const *T* &tuple, const *O* &ops, uint32_t index_base = 0)

Updates or inserts a tuple in the given space. The method works similar to *space_object:upsert()*.

If there is an existing tuple that matches the key fields of `tuple`, the request has the same effect as *update()* and the `ops` parameter is used. If there is no existing tuple that matches the key fields of `tuple`, the request has the same effect as *insert()* and the `tuple` parameter is used.

**Параметры**

- `tuple` – a tuple to insert.
- `ops` – parameters for the update operation, namely, `operator, field_identifier, value`.

- `index_base` – starting number to count fields in a tuple: `0` or `1`. Optional. Defaults to `0`.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to space_object:upsert({333, "upsert-insert", 0.0}, {{'=', 1, 'upsert-
↪update'}}) in Tarantool*/
uint32_t space_id = 512;
std::tuple tuple = std::make_tuple(333, "upsert-insert", 0.0);
std::tuple op1 = std::make_tuple("=", 1, "upsert-update");
rid_t f1 = conn.space[space_id].upsert(tuple, std::make_tuple(op1));
```

template<class `T`>
*rid_t* `delete_`(const *T* &key, uint32_t index_id = 0)

Deletes a tuple in the given space. The method works similar to *space_object:delete()* and searches for the tuple to delete against the primary index (`index_id = 0`) by default. In other words, `space[space_id].delete_()` equals to `space[space_id].index[0].delete_()`.

**Параметры**

- `key` – value to be matched against the index key.
- `index_id` – index ID. Optional. Defaults to `0`.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to space_object:delete(123) in Tarantool*/
uint32_t space_id = 512;
std::tuple key = std::make_tuple(123);
rid_t f1 = conn.space[space_id].delete_(key);
```

### Index class

class `Index` : *Space*

`Index` is a nested class of the *Space* class. It is a public wrapper to access the data-manipulation methods in the way similar to the Tarantool submodule *box.index*, like, `space[space_id].index[index_id].select()` and so on.

All the `Index` class methods listed below work in the following way:

- A method encodes the corresponding request in the MessagePack format and queues it in the output connection buffer to be sent later by one of *Connector's* methods, namely, *wait()*, *waitAll()*, or *waitAny()*.
- A method returns the request ID that is used to get the response by the *getResponce()* method. Refer to the *getResponce()* description to understand the response structure and how to read the requested data.

**Public methods**:

- *select()*
- *update()*
- *delete_()*

template<class T>
*rid_t* `select`(const *T* &key, uint32_t limit = UINT32_MAX, uint32_t offset = 0, IteratorType iterator = EQ)

This is an alternative to *space.select()*. The method searches for a tuple or a set of tuples in the given space against a particular index and works similar to *index_object:select()*.

**Параметры**

- `key` – value to be matched against the index key.
- `limit` – maximum number of tuples to select. Optional. Defaults to `UINT32_MAX`.
- `offset` – number of tuples to skip. Optional. Defaults to `0`.
- `iterator` – the type of iterator. Optional. Defaults to `EQ`.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to index_object:select({key}, {limit = 1}) in Tarantool*/
uint32_t space_id = 512;
uint32_t index_id = 1;
int key = 10;
uint32_t limit = 1;
auto i = conn.space[space_id].index[index_id];
rid_t select = i.select(std::make_tuple(key), limit, offset, iter);
```

template<class K, class T>
*rid_t* `update`(const *K* &key, const *T* &tuple)

This is an alternative to *space.update()*. The method updates a tuple in the given space but searches for the tuple against a particular index. The method works similar to *index_object:update()*.

The `tuple` parameter specifies an update operation, an identifier of the field to update, and a new field value. The set of available operations and the format of specifying an operation and a field identifier is the same as in Tarantool. Refer to the description of :doc:` </reference/reference_lua/box_index/update>` and example below for details.

**Параметры**

- `key` – value to be matched against the index key.
- `tuple` – parameters for the update operation, namely, `operator, field_identifier, value`.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to index_object:update(key, {{'=', 1, 'update' }, {'+', 2, 12}}) in␣
↪Tarantool*/
uint32_t space_id = 512;
uint32_t index_id = 1;
std::tuple key = std::make_tuple(10);
std::tuple op1 = std::make_tuple("=", 1, "update");
std::tuple op2 = std::make_tuple("+", 2, 12);
rid_t f1 = conn.space[space_id].index[index_id].update(key, std::make_tuple(op1,␣
↪op2));
```

template<class T>
*rid_t* delete_(const *T* &key)

This is an alternative to *space.delete_()*. The method deletes a tuple in the given space but searches for the tuple against a particular index. The method works similar to *index_object:delete()*.

**Параметры** key – value to be matched against the index key.

**Результат** a request ID

**Rtype** rid_t

**Possible errors:** none.

**Example:**

```
/* Equals to index_object:delete(123) in Tarantool*/
uint32_t space_id = 512;
uint32_t index_id = 1;
std::tuple key = std::make_tuple(123);
rid_t f1 = conn.space[space_id].index[index_id].delete_(key);
```

### 7.5.3 Node.js

Самый используемый драйвер для node.js – Node Tarantool driver. Он не входит в репозиторий Tarantool, его необходимо устанавливать отдельно. Проще всего установить его вместе с npm. Например, на Ubuntu, когда npm уже установлен, установка драйвера будет выглядеть следующим образом:

```
$ npm install tarantool-driver --global
```

Далее приводится пример полноценной программы на языке node.js, которая осуществляет вставку кортежа `[99999,'BB']` в спейс `space[999]` с помощью API для языка node.js. Перед запуском проверьте, что у экземпляра задан порт для *прослушивания* на `localhost:3301`, и в базе создан спейс `examples`, как *описано выше*. Чтобы запустить программу, сохраните код в файл с именем `example.rs` и выполните команду `node example.rs`. Программа установит соединение, используя определение спейса для этой цели, откроет сокет для соединения с экземпляром по `localhost:3301`, отправит *INSERT-запрос*, а затем – если всё хорошо – выдаст сообщение «Insert succeeded». Если Tarantool не запущен на `localhost` на прослушивание по порту = 3301, то программа выдаст сообщение об ошибке “Connect failed”. Если у *пользователя „guest“* нет прав на соединение, программа выдаст сообщение об ошибке «Auth failed». Если запрос вставки по какой-либо причине не сработает, например поскольку такой кортеж уже существует, то программа выдаст сообщение об ошибке «Insert failed».

```
var TarantoolConnection = require('tarantool-driver');
var conn = new TarantoolConnection({port: 3301});
var insertTuple = [99999, "BB"];
```

(continues on next page)

(продолжение с предыдущей страницы)

```
conn.connect().then(function() {
    conn.auth("guest", "").then(function() {
        conn.insert(999, insertTuple).then(function() {
            console.log("Insert succeeded");
            process.exit(0);
    }, function(e) { console.log("Insert failed");  process.exit(1); });
    }, function(e) { console.log("Auth failed");    process.exit(1); });
    }, function(e) { console.log("Connect failed"); process.exit(1); });
```

В этой программе мы привели пример использования лишь одного запроса. Для полноценной работы с Tarantool обратитесь к документации из репозитория драйвера для node.js.

### 7.5.4 Perl

The most commonly used Perl driver is tarantool-perl. It is not supplied as part of the Tarantool repository; it must be installed separately. The most common way to install it is by cloning from GitHub.

To avoid minor warnings that may appear the first time `tarantool-perl` is installed, start with installing some other modules that `tarantool-perl` uses, with CPAN, the Comprehensive Perl Archive Network:

```
$ sudo cpan install AnyEvent
$ sudo cpan install Devel::GlobalDestruction
```

Затем для установки самого `tarantool-perl`, выполните:

```
$ git clone https://github.com/tarantool/tarantool-perl.git tarantool-perl
$ cd tarantool-perl
$ git submodule init
$ git submodule update --recursive
$ perl Makefile.PL
$ make
$ sudo make install
```

Here is a complete Perl program that inserts `[99999,'BB']` into `space[999]` via the Perl API. Before trying to run, check that the server instance is listening at `localhost:3301` and that the space `examples` exists, as *described earlier*. To run, paste the code into a file named `example.pl` and say `perl example.pl`. The program will connect using an application-specific definition of the space. The program will open a socket connection with the Tarantool instance at `localhost:3301`, then send an *space_object:INSERT* request, then — if all is well — end without displaying any messages. If Tarantool is not running on `localhost` with *listen* port = 3301, the program will print "Connection refused".

```
#!/usr/bin/perl
#!/usr/bin/perl
use DR::Tarantool ':constant', 'tarantool';
use DR::Tarantool ':all';
use DR::Tarantool::MsgPack::SyncClient;

  host    => '127.0.0.1',                    # поиск Tarantool-сервера по адресу␣
↪localhost
  port    => 3301,                           # на порту 3301
  user    => 'guest',                        # имя пользователя; здесь же можно␣
↪добавить 'password=>...'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  spaces  => {
    999 => {                                  # определение спейса  space[999] ...
      name => 'examples',                     # имя спейса space[999] = 'examples'
      default_type => 'STR',                  # если тип поля в space[999] не задан, mo␣
↪= 'STR'
      fields => [ {                           # определение полей в спейсе space[999] ..
↪.
          name => 'field1', type => 'NUM' } ], # имя поля space[999].field[1]='field1',␣
↪тип ='NUM'
      indexes => {                            # определение индексов спейса space[999] .
↪..
        0 => {
          name => 'primary', fields => [ 'field1' ] } } } } );

$tnt->insert('examples' => [ 99999, 'BB' ]);
```

The example program uses field type names „STR“ and „NUM“ instead of *„string“ and „unsigned“*, due to a temporary Perl limitation.

The example program only shows one request and does not show all that's necessary for good practice. For that, please see the tarantool-perl repository.

### 7.5.5 PHP

tarantool-php is the official PHP connector for Tarantool. It is not supplied as part of the Tarantool repository and must be installed separately (see installation instructions in the connector's `README` file).

Далее приводится пример полноценной программы на языке PHP, которая осуществляет вставку кортежа `[99999,'BB']` в спейс `examples` с помощью API для языка PHP.

Перед запуском проверьте, что у экземпляра задан порт для *прослушивания* на `localhost:3301`, и в базе создан спейс `examples`, как *описано выше*.

Чтобы запустить программу, сохраните код в файл с именем `example.php` и выполните:

```
$ php -d extension=~/tarantool-php/modules/tarantool.so example.php
```

Программа откроет сокет для соединения с экземпляром по `localhost:3301`, отправит *INSERT-запрос*, а затем – если всё хорошо – выдаст сообщение «Insert succeeded».

Если такой кортеж уже существует, то программа выдаст сообщение об ошибке “Duplicate key exists in unique index „primary“ in space „examples““.

```
<?php
$tarantool = new Tarantool('localhost', 3301);

try {
    $tarantool->insert('examples', [99999, 'BB']);
    echo "Insert succeeded\n";
} catch (Exception $e) {
    echo $e->getMessage(), "\n";
}
```

The example program only shows one request and does not show all that's necessary for good practice. For that, please see tarantool/tarantool-php project at GitHub.

Besides, there is another community-driven tarantool-php GitHub project which includes an alternative connector written in pure PHP, an object mapper, a queue and other packages.

For Erlang, use the Erlang tarantool driver.

For R, use the tarantoolr connector.

## 7.6 Протокол

Бинарный протокол передачи данных в Tarantool был разработан с учетом потребностей асинхронного ввода-вывода для облегчения интеграции с прокси-серверами. Каждый клиентский запрос начинается с бинарного заголовка переменной длины. В заголовке указывается идентификатор и тип запроса, идентификатор экземпляра, номер записи в журнале и т.д.

Также в заголовке обязательно указывается длина запроса, что облегчает обмен данными с клиентом или прокси. Ответ на запрос посылается по мере готовности. В заголовке ответа указывается тот же идентификатор и тип запроса, что и в изначальном запросе. По идентификатору легко соотнести запрос с ответом, даже если ответы приходят в произвольном порядке.

Вдаваться в тонкости реализации бинарного протокола нужно, только если вы разрабатываете новый коннектор для Tarantool. В остальных случаях достаточно взять уже существующий коннектор для нужного вам языка программирования. Такие коннекторы позволяют без труда хранить в формате Tarantool структуры данных из разных языков. Полное описание бинарного протокола в Tarantool хранится в дереве исходного кода в виде аннотированных форм Бэкуса — Наура (BNF-диаграмм). Подробные примеры и диаграммы всех запросов и ответов можно найти в *описании бинарного протокола в Tarantool*.

## 7.7 Пример пакета данных

С помощью API Tarantool клиентские программы могут отправлять на адрес экземпляра пакеты с запросами и получать на них ответы. Пример ниже относится к клиентскому запросу `box.space[513]:insert{'A', 'BB'}`. Описания компонентов запроса, представленные в форме Бэкуса — Наура, вы найдете на странице о *бинарном протоколе Tarantool*.

| Компонент | Байт 0 | Байт 1 | Байт 2 | Байт 3 |
|---|---|---|---|---|
| код операции insert | 02 | | | |
| остальная часть заголовка | ... | ... | ... | ... |
| двузначное число: ID спейса | cd | 02 | 01 | |
| код кортежа | 21 | | | |
| однозначное число: количество полей = 2 | 92 | | | |
| односимвольная строка: поле[1] | a1 | 41 | | |
| двухсимвольная строка: поле[2] | a2 | 42 | 42 | |

Этот пакет можно отправить экземпляру Tarantool, а затем расшифровать ответ. Формат пакетов запроса и ответа также описан на странице о *бинарном протоколе Tarantool*. Однако проще и надежнее вызвать процедуру, которая сформирует готовый пакет с заданными параметрами. Например, она могла бы выглядеть так: `response = tarantool_routine("insert", 513, "A", "B");`. Для этого и существуют API для драйверов Perl, Python, PHP и т. д.

## 7.8 Настройка окружения для работы с примерами

В этой главе приводятся примеры того, как можно установить соединение с сервером Tarantool с помощью коннекторов для языков Perl, PHP, Python, Node.js и C. Примеры содержат фиксированные значения и будут корректно работать только при следующих условиях:

- Экземпляр (tarantool) запущен на локальной машине (`localhost = 127.0.0.1`), а прослушивание для него настроено на порту 3301 (`box.cfg.listen = '3301'`).
- В базе есть спейс `examples` с идентификатором 999 (`box.space.examples.id = 999`). Первичный индекс этого спейса построен по ключу числового типа (`box.space[999].index[0].parts[1].type = "unsigned"`).
- Пользователь „guest“ имеет разрешение на чтение и запись.

Можно легко соблюсти все условия, запустив экземпляр и выполнив следующий скрипт:

```
box.cfg{listen=3301}
box.schema.space.create('examples',{id=999})
box.space.examples:create_index('primary', {type = 'hash', parts = {1, 'unsigned'}})
box.schema.user.grant('guest','read,write','space','examples')
box.schema.user.grant('guest','read','space','_space')
```

## 7.9 Интерпретация возвращаемых значений

При работе с любым Tarantool-коннектором функции, вызванные Tarantool, возвращают значения в формате MsgPack. Для функций, вызываемых через API коннектора, формат возвращаемых значений следующий: скалярные значения возвращаются в виде кортежей (идентификатор типа в формате MsgPack, затем значение); все прочие (не скалярные) значения возвращаются в виде групп кортежей (идентификатор массива в формате MsgPack, затем скалярные значения). Если функция вызывается в рамках бинарного протокола (с помощью команды `eval`), а не через API коннектора, формат возвращаемых ею значений не меняется.

В примере ниже создается Lua-функция. Поскольку эту функцию будет вызывать внешний пользователь *„guest“ user*, необходимо с помощью *grant* настроить права на исполнение. Функция возвращает пустой массив, строку-скаляр, два логических значения и короткое целое число. Значения соответствуют приведенным в таблице *стандартных типов в MsgPack-кодировке*.

```
tarantool> box.cfg{listen=3301}
2016-03-03 18:45:52.802 [27381] main/101/interactive I> ready to accept requests
---
...
tarantool> function f() return {},'a',false,true,127; end
---
...
tarantool> box.schema.func.create('f')
---
...
tarantool> box.schema.user.grant('guest','execute','function','f')
---
...
```

Следующая программа на C вызывает эту функцию. Хотя в примере приводится код на C, результат будет одинаковым, на каком бы языке ни была написана программа: Perl, PHP, Python, Go или Java.

```
#include <stdio.h>
#include <stdlib.h>
#include <tarantool/tarantool.h>
#include <tarantool/tnt_net.h>
#include <tarantool/tnt_opt.h>
void main() {
  struct tnt_stream *tnt = tnt_net(NULL);                /* Настройка */
  tnt_set(tnt, TNT_OPT_URI, "localhost:3301");
   if (tnt_connect(tnt) < 0) {                           /* Соединение */
       printf("Connection refused\n");
       exit(-1);
   }
   struct tnt_stream *arg; arg = tnt_object(NULL);       /* Формирование запроса */
   tnt_object_add_array(arg, 0);
   struct tnt_request *req1 = tnt_request_call(NULL);    /* Вызов функции f() */
   tnt_request_set_funcz(req1, "f");
   uint64_t sync1 = tnt_request_compile(tnt, req1);
   tnt_flush(tnt);                                       /* Отправка запроса */
   struct tnt_reply reply;  tnt_reply_init(&reply);      /* Получение ответа */
   tnt->read_reply(tnt, &reply);
   if (reply.code != 0) {
     printf("Call failed %lu.\n", reply.code);
     exit(-1);
   }
   const unsigned char *p= (unsigned char*)reply.data; /* Вывод ответа */
   while (p < (unsigned char *) reply.data_end)
   {
     printf("%x ", *p);
     ++p;
   }
   printf("\n");
   tnt_close(tnt);                                       /* Завершение */
   tnt_stream_free(arg);
   tnt_stream_free(tnt);
}
```

По завершении программа выведет на экран следующие значения:

```
dd 0 0 0 5 90 91 a1 61 91 c2 91 c3 91 7f
```

Первые пять байт, `dd 0 0 0 5`, представляют собой фрагмент данных в формате MsgPack и обозначают 32-битный заголовок массива со значением 5 (см. спецификацию формата MsgPack). Остальные значения описаны в таблице *стандартных типов в MsgPack-кодировке*.

ГЛАВА 8

# Tarantool Enterprise Edition

Данное руководство посвящено Enterprise-версии продукта Tarantool, который сочетает в себе сервер приложений Lua и отказоустойчивую распределенную СУБД.

Enterprise-версия предлагает дополнительные возможности по разработке и эксплуатации кластерных приложений, например:

- Статическая сборка для автономных Linux-систем.
- Модуль интеграции с OpenLDAP.
- *Журнал аудита безопасности*.
- Подключения к корпоративным базам данных: Oracle и любым СУБД с интерфейсом ODBC (например, MySQL, Microsoft SQL Server).
- *Шифрование трафика* с помощью SSL.
- *Сжатие кортежей*.
- *Смена схемы данных в фоновом режиме*.

Enterprise-версия распространяется в форме SDK, который включает следующие ключевые компоненты:

- Исполняемый файл Tarantool EE binary, который может работать с *цетрализованной конфигурацией*.
- Расширенная Enterprise-версия утилиты *tt*.
- *Tarantool Cluster Manager* – визуальный веб-инструмент для управления кластерами Tarantool.

## 8.1 Настройка

This chapter explains how to download and set up Tarantool Enterprise Edition and run a sample application provided with it.

### 8.1.1 Системные требования

Ниже представлены рекомендуемые системные требования для запуска Tarantool Enterprise.

#### Требования к аппаратному обеспечению

Чтобы обеспечить полную отказоустойчивость системы распределенного хранения данных, необходимы как минимум **три** физических компьютера или виртуальных сервера.

Для целей тестирования или разработки систему можно развернуть, используя меньшее количество серверов. Тем не менее, не рекомендуется использовать такие конфигурации в производственной среде.

#### Требования к программному обеспечению

1. Tarantool Enterprise поддерживает операционные системы **Red Hat Enterprise Linux** и **CentOS** версии 7.5 и выше.

   ---

   **Примечание:** Tarantool Enterprise может работать на других дистрибутивах Linux на основе `systemd`, но тестирование на них не проводится, поэтому корректная работа не гарантирована.

   ---

2. Требуется `glibc` версии 2.17-260.el7_6.6 и выше. Необходимо проверить текущую версию и обновить в случае необходимости:

   ```
   $ rpm -q glibc
   glibc-2.17-196.el7_4.2
   $ yum update glibc
   ```

#### Сетевые требования

Здесь и далее по тексту под **серверами хранения данных** или **серверами Tarantool** понимаются компьютеры, которые используются для хранения и обработки данных, а под **сервером администрирования** понимается компьютер, с помощью которого оператор устанавливает и настраивает систему.

Кластер Tarantool работает по принципам полносвязной топологии (full mesh topology), поэтому все серверы Tarantool должны поддерживать прием и передачу данных по TCP и UDP на всех портах, которые используются экземплярами кластера (см. настройки `advertise_uri: <host>:<port>` и `config: advertise_uri: '<host>:<port>'` в файле `/etc/tarantool/conf.d/*.yml` для каждого экземпляра). Например:

```
# /etc/tarantool/conf.d/*.yml

myapp.s2-replica:
  advertise_uri: localhost:3305 # this is a TCP/UDP port
  http_port: 8085
```

(continues on next page)

(продолжение с предыдущей страницы)

```
all:
  ...
  hosts:
    storage-1:
      config:
        advertise_uri: 'vm1:3301' # this is a TCP/UDP port
        http_port: 8081
```

Чтобы настроить удаленный мониторинг или подключиться по административной консоли, сервер администрирования должен иметь доступ к следующим TCP-портам на серверах Tarantool:

- 22 – чтобы использовать SSH-протокол;
- ports specified in instance configuration to monitor the HTTP-metrics.

Кроме того, рекомендуется применить следующие настройки для `sysctl` на всех серверах Tarantool:

```
$ # TCP KeepAlive setting
$ sysctl -w net.ipv4.tcp_keepalive_time=60
$ sysctl -w net.ipv4.tcp_keepalive_intvl=5
$ sysctl -w net.ipv4.tcp_keepalive_probes=5
```

Эта необязательная настройка сетевого стека Linux помогает ускорить решение проблем с сетевым подключением при физическом отказе сервера. Для достижения максимальной производительности может также потребоваться настройка других параметров сетевого стека, которые не относятся к СУБД Tarantool. Для получения дополнительной информации обратитесь к разделу Руководство по оптимизации сетевой производительности (Network Performance Tuning Guide) в пользовательской документации по RHEL7.

### 8.1.2 Комплектация

The latest release packages of Tarantool Enterprise are available in the customer zone at Tarantool website. Please contact `support@tarantool.io` for access.

Каждый пакет представляет собой архив `tar + gzip` и включает в себя следующие компоненты и функции:

- Static Tarantool binary for simplified deployment in Linux environments.
- `tt` command-line utility that provides a unified command-line interface for managing Tarantool-based applications. See *tt CLI utility* for details.
- Tarantool Cluster Manager – a web-based interface for managing Tarantool EE clusters. See *Tarantool Cluster Manager* for details.
- Selection of open and closed source modules.
- Sample application walking you through all included modules

Содержимое архива:

- `tarantool` – основной исполняемый файл Tarantool.
- `tt` command-line utility.
- `tcm` is the Tarantool Cluster Manager executable.

- `tarantoolctl` – служебный скрипт для установки вспомогательных модулей и подключения к административной консоли.

  **Важно:** `tarantoolctl` is deprecated in favor of the *tt CLI utility*.

- `examples/` – директория, которая содержит примеры приложений:
  - `pg_writethrough_cache/` – приложение, которое наглядно показывает, как Tarantool может кэшировать данные, например, записанные в базу данных PostgreSQL;
  - `ora_writebehind_cache/` – приложение, которое наглядно показывает, как Tarantool может кэшировать вставки и ставить их в очередь, например, в базу данных Oracle;
  - `docker/` – приложение, предназначенное для простой упаковки в Docker-контейнер;
- `rocks/` – директория, которая содержит набор дополнительных модулей с открытым и закрытым исходным кодом, включенных в дистрибутив в качестве автономного репозитория сторонних библиотек. Для получения более подробной информации см. *справочник по сторонним библиотекам*.
- `templates/` – директория, которая содержит файлы шаблонов для среды разработки приложений.
- `deprecated/` – это набор модулей, которые больше не поддерживаются:
  - `vshard-zookeeper-orchestrator` – это приложение на Python для запуска orchestrator,
  - файлы `zookeeper-scm` представляют собой модули для интеграции с ZooKeeper (требуются библиотеки `usr/`).

### 8.1.3 Установка

Готовый архив `tar + gzip` необходимо загрузить на сервер и распаковать:

```
$ tar xvf tarantool-enterprise-sdk-<version>.tar.gz
```

Дополнительная установка не требуется, поскольку распакованные бинарные файлы практически готовы к работе. Перейдите в каталог с бинарными файлами (`tarantool-enterprise`) и добавьте их в путь для поиска исполняемых файлов, запустив скрипт из дистрибутива:

```
$ source ./env.sh
```

Убедитесь, что вы можете запускать данный скрипт, а также что файл со скриптом является исполняемым. В противном случае задайте разрешения в помощью команд `chmod` и `chown`.

Затем настройте среду разработки, как описано в руководстве для разработчика.

## 8.2 Инструкции по повышению безопасности

This guide explains how to enhance security in your Tarantool Enterprise Edition's cluster using built-in features and provides general recommendations on security hardening. If you need to perform a security audit of a Tarantool Enterprise cluster, refer to the *security checklist*.

Tarantool Enterprise Edition does not provide a dedicated API for security control. All the necessary configurations can be done via an administrative console or initialization code.

В Tarantool Enterprise есть следующие встроенные средства безопасности:

- *authentication*
- *access control*
- *audit log*
- *traffic encryption*

### 8.2.1 Аутентификация

Tarantool Enterprise поддерживает аутентификацию на основе паролей и допускает два типа соединений:

- Via an *administrative console*.
- Over a binary port for read and write operations and procedure invocation.

For more information on authentication and connection types, see the *Security* section of the Tarantool manual.

Кроме того, Tarantool предоставляет следующие функциональные возможности:

- *Sessions* – states which associate connections with users and make Tarantool API available to them after authentication.
- Authentication *triggers*, which execute actions on authentication events.
- Third-party (external) authentication protocols and services such as LDAP or Active Directory – supported in the web interface, but unavailable on the binary-protocol level.

### 8.2.2 Управление доступом

Для администраторов Tarantool Enterprise предоставляет средства предотвращения несанкционированного доступа к базе данных и к определенным функциям.

Tarantool различает:

- different users (guests and administrators)
- privileges associated with users
- roles (containers for privileges) granted to users

The following system spaces are used to store users and privileges:

- The `_user` space to store usernames and hashed passwords for authentication.
- The `_priv` space to store privileges for access control.

For more information, see the *Access control* section.

Users who create **objects** (spaces, indexes, users, roles, sequences, and functions) in the database become their **owners** and automatically acquire privileges for what they create. For more information, see the *Owners and privileges* section.

### 8.2.3 Журнал аудита

В Tarantool Enterprise есть встроенный журнал аудита, в котором записываются такие события, как:

- authentication successes and failures
- connection closures
- creation, removal, enabling, and disabling of users
- changes of passwords, privileges, and roles
- denials of access to database objects

The audit log contains:

- timestamps
- usernames of users who performed actions
- event types (for example, `user_create`, `user_enable`, `disconnect`)
- descriptions

You can configure the following audit log options:

- *audit_log.to* – enable audit logging and define the log location (file, pipe, or syslog). The option is similar to the *log*.
- *audit_log.nonblock* – specify the logging behavior if the system is not ready to write. The option is similar to the *log_nonblock*.

Для получения дополнительной информации о журналировании см. следующие разделы:

- the *Logs* section
- the *Logging* section in the configuration reference
- the *Tarantool audit module* topic

Права доступа к файлам журнала можно настроить, как для любого другого объекта файловой системы Unix – через `chmod`.

### 8.2.4 Рекомендации по повышению безопасности

В этом разделе даны рекомендации, которые могут помочь вам повысить безопасность кластера.

#### Encrypting traffic

Since version 2.10.0, Tarantool Enterprise Edition has built-in support for using SSL to encrypt the client-server communications over binary connections, that is, between Tarantool instances in a cluster. For details on enabling SSL encryption, see the *Securing connections with SSL* section of this guide.

In case the built-in encryption is not set for particular connections, consider the following security recommendations:

- настроить туннелирование соединения или
- зашифровать сами данные, которые хранятся в базе.

For more information on data encryption, see the *crypto module reference*.

The HTTP server module provided by rocks does not support the HTTPS protocol. To set up a secure connection for a client (e.g., REST service), consider hiding the Tarantool instance (router if it is a cluster of instances) behind an Nginx server and setting up an SSL certificate for it.

To make sure that no information can be intercepted „from the wild“, run nginx on the same physical server as the instance and set up their communication over a Unix socket. For more information, see the *socket module reference*.

#### Настройка брандмауэра

To protect the cluster from any unwanted network activity „from the wild“, configure the firewall on each server to allow traffic on ports listed in *Network requirements*.

Если вы используете статические IP-адреса, повторно внесите их в белый список на каждом сервере, поскольку кластер работает на принципах полносвязной топологии (full mesh topology). Рекомендуется внести в черный список всех остальные адреса на всех серверах, кроме роутера (работающего за сервером Nginx).

Tarantool Enterprise не предоставляет защиту от DoS-атак или DDoS-атак. Для этих целей рекомендуется использовать сторонние программы.

#### Целостность данных

Tarantool Enterprise Edition does not keep checksums or provide the means to control data integrity. However, it ensures data persistence using a write-ahead log, regularly snapshots the entire data set to disk, and checks the data format whenever it reads the data back from the disk. For more information, see the *Data persistence* section.

## 8.3 Аудит безопасности

This document will help you audit the security of a Tarantool cluster. It explains certain security aspects, their rationale, and the ways to check them. For details on how to configure Tarantool Enterprise Edition and its infrastructure for each aspect, refer to the *security hardening guide*.

### 8.3.1 Шифрование внешнего трафика iproto

Tarantool uses the *iproto binary protocol* for replicating data between instances and also in the connector libraries.

Since version 2.10.0, the Enterprise Edition has the built-in support for using SSL to encrypt the client-server communications over binary connections. For details on enabling SSL encryption, see the *Securing connections with SSL* section of this document.

In case the built-in encryption is not enabled, we recommend using VPN to secure data exchange between data centers.

### 8.3.2 Закрытые порты iproto

Если в кластере Tarantool не используется iproto для внешних запросов, подключение к портам iproto должно разрешаться только между экземплярами Tarantool.

Подробнее о настройке портов для iproto см. в разделе `advertise_uri` в документации Cartridge.

### 8.3.3 Механизм HTTPS termination

Экземпляр Tarantool может принимать HTTP-подключения от внешних источников или при доступе к веб-интерфейсу администратора. Все такие подключения должны проходить через веб-сервер с HTTPS, работающий на том же хосте (например, nginx). Это требование относится как к виртуальным, так и к физическим хостам. Проведение HTTP-трафика через несколько отдельных хостов с механизмом HTTPS termination недостаточно безопасно.

### 8.3.4 Закрытые HTTP-порты

Tarantool accepts HTTP connections on a specific port. It must be only available on the same host for nginx to connect to it.

Убедитесь, что настроенный HTTP-порт закрыт, а HTTPS-порт ( по умолчанию 443) открыт.

### 8.3.5 Ограниченный доступ к административной консоли

Модуль *console* дает возможность подключиться к рабочему экземпляру и запускать пользовательский код Lua. Это полезная возможность для разработчиков и администраторов. В следующих примерах показано, как открыть соединение по TCP-порту и на UNIX-сокете.

```
console.listen(<port number>)
console.listen('/var/lib/tarantool/socket_name.sock')
```

Открывать административную консоль через TCP-порт всегда небезопасно. Убедитесь, что в коде нет вызовов формата `console.listen(<port_number>)`.

При подключении через сокет требуется право write для директории `/var/lib/tarantool`. Убедитесь, что оно выдано только пользователю `tarantool`.

### 8.3.6 Ограничения для пользователя guest

Connecting to the instance with `tt connect` or `tarantoolctl connect` without user credentials (under the `guest` user) must be disabled.

Есть два способа проверить эту уязвимость:

- Убедитесь, что в исходном коде не предоставляются права доступа пользователю `guest`. Этот участок кода может выглядеть так:

```
box.schema.user.grant('guest',
    'read,write',
    'universe',
    nil, { if_not_exists = true }
)
```

Помимо сооветствующего фрагмента кода, ищите все вхождения значения `'universe'`.

- Try connecting with `tt connect` to each Tarantool node.

For more details, refer to the documentation on *access control*.

### 8.3.7 Авторизация в веб-интерфейсе

Using the web interface must require logging in with a username and password.

### 8.3.8 Запуск под пользователем tarantool

Все экземпляры Tarantool должны работать под пользователем `tarantool`.

### 8.3.9 Ограничение доступа для пользователя tarantool

Пользователь `tarantool` не должен иметь права `sudo`. Кроме того, у него не должно быть пароля, чтобы не допустить входа через SSH или `su`.

### 8.3.10 Хранение двух и более снимков данных

Для надежности резервного копирования в экземпляре Tarantool должны храниться минимум два последних снимка данных. Не забудьте проверить каждый экземпляр.

The *snapshot_count* value determines the number of kept snapshots. Configuration values are primarily set in the configuration files but can be overridden with environment variables and command-line arguments. So, it's best to check both the values in the configuration files and the actual values using the console:

```
tarantool> box.cfg.checkpoint_count
---
- 2
```

### 8.3.11 Включен журнал упреждающей записи (WAL)

Tarantool фиксирует все входящие данные в журнале упреждающей записи (WAL). WAL должен быть включен, чтобы в случае перезапуска экземпляра эти данные можно было восстановить.

Secure values of the *wal.mode* configuration option are `write` and `fsync`:

```
wal:
  dir: 'var/lib/{{ instance_name }}/wals'
  mode: 'write'
```

An exclusion from this requirement is when the instance is processing data, which can be freely rejected - for example, when Tarantool is used for caching. In this case, WAL can be disabled to reduce i/o load.

### 8.3.12 Уровень ведения журнала: INFO или выше

Уровень ведения журнала должен быть 5 (`INFO`), 6 (`VERBOSE`) или 7 (`DEBUG`). Тогда в случае нарушения безопасности в журналах приложения будет достаточно информации для расследования инцидента.

```
tarantool> box.cfg.log_level
---
- 5
```

Полный список существующих уровней см. в разделе *справки по log_level*.

### 8.3.13 Ведение журнала с помощью journald

В Tarantool для ведения журнала следует использовать `journald`.

## 8.4 Tuple compression

Tuple compression, introduced in Tarantool Enterprise Edition 2.10.0, aims to save memory space. Typically, it decreases the volume of stored data by 15%. However, the exact volume saved depends on the type of data.

The following compression algorithms are supported:

- lz4
- zstd
- zlib (since *2.11.0*)

To learn about the performance costs of each algorithm, check *Tuple compression performance*.

Tarantool doesn't compress tuples themselves, just the fields inside these tuples. You can only compress non-indexed fields. Compression works best when JSON is stored in the field.

---

**Примечание:** The *compress* module provides the API for compressing and decompressing data.

---

### 8.4.1 Enabling compression for a new space

First, create a space:

```
box.schema.space.create('bands')
```

Then, create an index for this space, for example:

```
box.space.bands:create_index('primary', {parts = {{1, 'unsigned'}}})
```

Create a format to declare field names and types. In the example below, the `band_name` and `year` fields have the `zstd` and `lz4` compression formats, respectively. The first field (`id`) has the index, so it cannot be compressed.

```
box.space.bands:format({
           {name = 'id', type = 'unsigned'},
           {name = 'band_name', type = 'string', compression = 'zstd'},
           {name = 'year', type = 'unsigned', compression = 'lz4'}
       })
```

Now, the new tuples that you add to the space `bands` will be compressed. When you read a compressed tuple, you do not need to decompress it back yourself.

### 8.4.2 Checking which fields are compressed

To check which fields in a space are compressed, run *space_object:format()* on the space. If a field is compressed, the format includes the compression algorithm, for example:

```
tarantool> box.space.bands:format()
    ---
    - [{'name': 'id', 'type': 'unsigned'},
       {'type': 'string', 'compression': 'zstd', 'name': 'band_name'},
       {'type': 'unsigned', 'compression': 'lz4', 'name': 'year'}]
    ...
```

### 8.4.3 Enabling compression for existing spaces

You can enable compression for existing fields. All the tuples added after that will have this field compressed. However, this doesn't affect the tuples already stored in the space. You need to make the snapshot and restart Tarantool to compress the existing tuples.

Here's an example of how to compress existing fields:

1. Create a space without compression and add several tuples:

   ```
   box.schema.space.create('bands')

   box.space.bands:format({
       { name = 'id', type = 'unsigned' },
       { name = 'band_name', type = 'string' },
       { name = 'year', type = 'unsigned' }
   })

   box.space.bands:create_index('primary', { parts = { 'id' } })

   box.space.bands:insert { 1, 'Roxette', 1986 }
   box.space.bands:insert { 2, 'Scorpions', 1965 }
   box.space.bands:insert { 3, 'Ace of Base', 1987 }
   box.space.bands:insert { 4, 'The Beatles', 1960 }
   ```

2. Suppose that you want fields 2 and 3 to be compressed from now on. To enable compression, change the format as follows:

   ```
   local new_format = box.space.bands:format()

   new_format[2].compression = 'zstd'
   ```

(continues on next page)

(продолжение с предыдущей страницы)

```
new_format[3].compression = 'lz4'

box.space.bands:format(new_format)
```

From now on, all the tuples that you add to the space have fields 2 and 3 compressed.

3. To finalize the change, create a snapshot by running *box.snapshot()* and restart Tarantool. As a result, all old tuples will also be compressed in memory during recovery.

---

**Примечание:** *space:upgrade()* provides the ability to enable compression and update the existing tuples in the background. To achieve this, you need to pass a new space format in the `format` argument of `space:upgrade()`.

---

### 8.4.4 Tuple compression performance

Below are the results of a synthetic test that illustrate how tuple compression affects performance. The test was carried out on a simple Tarantool space containing 100,000 tuples, each having a field with a sample JSON roughly 600 bytes large. The test compared the speed of running `select` and `replace` operations on uncompressed and compressed data as well as the overall data size of the space. Performance is measured in requests per second.

| Compression type | `select`, RPS | `replace`, RPS | Space size, bytes |
|---|---|---|---|
| None | 4,486k | 1,109k | 41,168,548 |
| `zstd` | 308k | 26k | 21,368,548 |
| `lz4` | 1,765k | 672k | 25,268,548 |
| `zlib` | 325k | 107k | 20,768,548 |

#### Module compress

**Since:** *2.11.0*

The `compress` module provides a set of submodules for compressing and decompressing data using different algorithms:

#### Submodule compress.zlib

#### Overview

The `compress.zlib` submodule provides the ability to compress and decompress data using the zlib algorithm. You can use the `zlib` compressor as follows:

1. Create a compressor instance using the *compress.zlib.new()* function:

```
local zlib_compressor = require('compress.zlib').new()
-- or --
local zlib_compressor = require('compress').zlib.new()
```

Optionally, you can pass compression options (*zlib_opts*) specific for `zlib`:

```
local zlib_compressor = require('compress.zlib').new({
    level = 5,
    mem_level = 5,
    strategy = 'filtered'
})
```

2. To compress the specified data, use the *compress()* method:

```
compressed_data = zlib_compressor:compress('Hello world!')
```

3. To decompress data, call *decompress()*:

```
decompressed_data = zlib_compressor:decompress(compressed_data)
```

### API Reference

| **Functions** | |
|---|---|
| *compress.zlib.new()* | Create a `zlib` compressor instance. |
| **Objects** | |
| *zlib_compressor* | A `zlib` compressor instance. |
| *zlib_opts* | Configuration options of the `zlib` compressor. |

### compress.zlib.new()

`compress.zlib.new(`[*zlib_opts*]`)`

Create a `zlib` compressor instance.

**Параметры**

- `options` (*table*) – `zlib` compression options (see *zlib_opts*)

**Return** a new `zlib` compressor instance (see *zlib_compressor*)

**Rtype** userdata

**Example**

```
local zlib_compressor = require('compress.zlib').new({
    level = 5,
    mem_level = 5,
    strategy = 'filtered'
})
```

### zlib_compressor

object zlib_compressor

A compressor instance that exposes the API for compressing and decompressing data using the `zlib` algorithm. To create the `zlib` compressor, call *compress.zlib.new()*.

zlib_compressor:compress(*data*)

Compress the specified data.

**Параметры**

- `data` (*string*) – data to be compressed

**Return** compressed data

**Rtype** *string*

**Example**

```
compressed_data = zlib_compressor:compress('Hello world!')
```

zlib_compressor:decompress(*data*)

Decompress the specified data.

**Параметры**

- `data` (*string*) – data to be decompressed

**Return** decompressed data

**Rtype** *string*

**Example**

```
decompressed_data = zlib_compressor:decompress(compressed_data)
```

### zlib_opts

object zlib_opts

Configuration options of the *zlib_compressor*. These options can be passed to the *compress.zlib.new()* function.

**Example**

```
local zlib_compressor = require('compress.zlib').new({
    level = 5,
    mem_level = 5,
    strategy = 'filtered'
})
```

zlib_opts.level

Specifies the `zlib` compression level that enables you to adjust the compression ratio and speed. The lower level improves the compression speed at the cost of compression ratio.

Default: 6

Minimum: 0 (no compression)

Maximum: 9

`zlib_opts.mem_level`
: Specifies how much memory is allocated for the `zlib` compressor. The larger value improves the compression speed and ratio.

  Default: 8
  Minimum: 1
  Maximum: 9

`zlib_opts.strategy`
: Specifies the compression strategy. The possible values:

  - `default` - for normal data.
  - `huffman_only` - forces Huffman encoding only (no string match). The fastest compression algorithm but not very effective in compression for most of the data.
  - `filtered` - for data produced by a filter or predictor. Filtered data consists mostly of small values with a somewhat random distribution. This compression algorithm is tuned to compress them better.
  - `rle` - limits match distances to one (run-length encoding). `rle` is designed to be almost as fast as `huffman_only` but gives better compression for PNG image data.
  - `fixed` - prevents the use of dynamic Huffman codes and provides a simpler decoder for special applications.

### Submodule compress.zstd

#### Overview

The `compress.zstd` submodule provides the ability to compress and decompress data using the zstd algorithm. You can use the `zstd` compressor as follows:

1. Create a compressor instance using the *compress.zstd.new()* function:

   ```
   local zstd_compressor = require('compress.zstd').new()
   -- or --
   local zstd_compressor = require('compress').zstd.new()
   ```

   Optionally, you can pass compression options (*zstd_opts*) specific for `zstd`:

   ```
   local zstd_compressor = require('compress.zstd').new({
       level = 5
   })
   ```

2. To compress the specified data, use the *compress()* method:

   ```
   compressed_data = zstd_compressor:compress('Hello world!')
   ```

3. To decompress data, call *decompress()*:

   ```
   decompressed_data = zstd_compressor:decompress(compressed_data)
   ```

### API Reference

| Functions | |
|---|---|
| *compress.zstd.new()* | Create a `zstd` compressor instance. |
| **Objects** | |
| *zstd_compressor* | A `zstd` compressor instance. |
| *zstd_opts* | Configuration options of the `zstd` compressor. |

### compress.zstd.new()

`compress.zstd.new([zstd_opts])`

Create a `zstd` compressor instance.

**Параметры**

- `options` (*table*) – `zstd` compression options (see *zstd_opts*)

**Return** a new `zstd` compressor instance (see *zstd_compressor*)

**Rtype** userdata

**Example**

```
local zstd_compressor = require('compress.zstd').new({
    level = 5
})
```

### zstd_compressor

`object zstd_compressor`

A compressor instance that exposes the API for compressing and decompressing data using the `zstd` algorithm. To create the `zstd` compressor, call *compress.zstd.new()*.

`zstd_compressor:compress(data)`

Compress the specified data.

**Параметры**

- `data` (*string*) – data to be compressed

**Return** compressed data

**Rtype** *string*

**Example**

```
compressed_data = zstd_compressor:compress('Hello world!')
```

`zstd_compressor:decompress(data)`

Decompress the specified data.

**Параметры**

- `data` (*string*) – data to be decompressed

**Return** decompressed data

**Rtype** *string*

**Example**

```
decompressed_data = zstd_compressor:decompress(compressed_data)
```

### zstd_opts

object zstd_opts

Configuration options of the *zstd_compressor*. These options can be passed to the *compress.zstd.new()* function.

**Example**

```
local zstd_compressor = require('compress.zstd').new({
    level = 5
})
```

zstd_opts.level

Specifies the `zstd` compression level that enables you to adjust the compression ratio and speed. The lower level improves the compression speed at the cost of compression ratio. For example, you can use level 1 if speed is most important and level 22 if size is most important.

Default: 3

Minimum: -131072

Maximum: 22

---

**Примечание:** Assigning 0 to `level` resets its value to the default (3).

---

## Submodule compress.lz4

### Overview

The `compress.lz4` submodule provides the ability to compress and decompress data using the lz4 algorithm. You can use the `lz4` compressor as follows:

1. Create a compressor instance using the *compress.lz4.new()* function:

   ```
   local lz4_compressor = require('compress.lz4').new()
   -- or --
   local lz4_compressor = require('compress').lz4.new()
   ```

   Optionally, you can pass compression options (*lz4_opts*) specific for `lz4`:

   ```
   local lz4_compressor = require('compress.lz4').new({
       acceleration = 1000,
       decompress_buffer_size = 2097152
   })
   ```

2. To compress the specified data, use the *compress()* method:

```
compressed_data = lz4_compressor:compress('Hello world!')
```

3. To decompress data, call *decompress()*:

```
decompressed_data = lz4_compressor:decompress(compressed_data)
```

### API Reference

| Functions | |
|---|---|
| *compress.lz4.new()* | Create a `lz4` compressor instance. |
| **Objects** | |
| *lz4_compressor* | A `lz4` compressor instance. |
| *lz4_opts* | Configuration options of the `lz4` compressor. |

### compress.lz4.new()

`compress.lz4.new([lz4_opts])`
Create a `lz4` compressor instance.

**Параметры**

- `options` (*table*) – `lz4` compression options (see *lz4_opts*)

**Return** a new `lz4` compressor instance (see *lz4_compressor*)

**Rtype** userdata

**Example**

```
local lz4_compressor = require('compress.lz4').new({
    acceleration = 1000,
    decompress_buffer_size = 2097152
})
```

### lz4_compressor

`object lz4_compressor`
A compressor instance that exposes the API for compressing and decompressing data using the `lz4` algorithm. To create the `lz4` compressor, call *compress.lz4.new()*.

`lz4_compressor:compress(data)`
Compress the specified data.

**Параметры**

- `data` (*string*) – data to be compressed

**Return** compressed data

**Rtype** *string*

**Example**

```
compressed_data = lz4_compressor:compress('Hello world!')
```

lz4_compressor:decompress(*data*)
Decompress the specified data.

**Параметры**

- data (*string*) – data to be decompressed

**Return** decompressed data

**Rtype** *string*

**Example**

```
decompressed_data = lz4_compressor:decompress(compressed_data)
```

### lz4_opts

object lz4_opts
Configuration options of the *lz4_compressor*. These options can be passed to the *compress.lz4.new()* function.

**Example**

```
local lz4_compressor = require('compress.lz4').new({
    acceleration = 1000,
    decompress_buffer_size = 2097152
})
```

lz4_opts.acceleration
Specifies the acceleration factor that enables you to adjust the compression ratio and speed. The higher acceleration factor increases the compression speed but decreases the compression ratio.

Default: 1
Maximum: 65537
Minimum: 1

lz4_opts.decompress_buffer_size
Specifies the decompress buffer size (in bytes). If the size of decompressed data is larger than this value, the compressor returns an error on decompression.

Default: 1048576

## 8.5 WAL extensions

WAL extensions available in Tarantool Enterprise Edition allow you to add auxiliary information to each *write-ahead log* record. For example, you can enable storing an old and new tuple for each CRUD operation performed. This information might be helpful for implementing a CDC (Change Data Capture) utility that transforms a data replication stream.

See also: *Configure the write-ahead log*.

### 8.5.1 Configuration

WAL extensions are disabled by default. To configure them, use the *wal.ext.** configuration options. Inside the `wal.ext` block, you can enable storing old and new tuples as follows:

- To store old and new tuples in a write-ahead log for all spaces, set the *wal.ext.old* and *wal.ext.new* options to `true`:

```
ext:
  new: true
  old: true
```

- To adjust these options for specific spaces, specify the *wal.ext.spaces* option:

```
wal:
  ext:
    old: true
    new: true
    spaces:
      space1:
        old: false
      space2:
        new: false
```

  The configuration for specific spaces has priority over the configuration in the `wal.ext.new` and `wal.ext.old` options. It means that only new tuples are added to the log for `space1` and only old tuples for `space2`.

Note that records with additional fields are *replicated* as follows:

- If a replica doesn't support the extended format configured on a master, auxiliary fields are skipped.
- If a replica and master have different configurations for WAL records, the master's configuration is ignored.

### 8.5.2 Example

The table below demonstrates how write-ahead log records might look for the specific *CRUD operations* if storing old and new tuples is enabled for the `bands` space.

| Operation | Example | WAL information |
|---|---|---|
| insert | `bands:insert{4, 'The Beatles', 1960}` | **new_tuple**: [4, „The Beatles“, 1960]<br>tuple: [4, „The Beatles“, 1960] |
| delete | `bands:delete{4}` | key: [4]<br>**old_tuple**: [4, „The Beatles“, 1960] |
| update | `bands:update({2}, {{'=', 2, 'Pink Floyd'}})` | **new_tuple**: [2, „Pink Floyd“, 1965]<br>**old_tuple**: [2, „Scorpions“, 1965]<br>key: [2]<br>tuple: [[„=“, 2, „Pink Floyd“]] |
| upsert | `bands:upsert({2, 'Pink Floyd', 1965}, {{'=', 2, 'The Doors'}})` | **new_tuple**: [2, „The Doors“, 1965]<br>**old_tuple**: [2, „Pink Floyd“, 1965]<br>operations: [[„=“, 2, „The Doors“]]<br>tuple: [2, „Pink Floyd“, 1965] |
| replace | `bands:replace{1, 'The Beatles', 1960}` | **old_tuple**: [1, „Roxette“, 1986]<br>**new_tuple**: [1, „The Beatles“, 1960]<br>tuple: [1, „The Beatles“, 1960] |

Storing both old and new tuples is especially useful for the `update` operation because a write-ahead log record contains only a key value.

**Примечание:** You can use the *tt cat* command to see the contents of a write-ahead log.

## 8.6 Read views

A read view is an in-memory snapshot of the entire database that isn't affected by future *data modifications*. Read views provide access to database spaces and their indexes and enable you to retrieve data using the same `select` and `pairs` operations.

Read views can be used to make complex analytical queries. This reduces the load on the main database and improves RPS for a single Tarantool instance.

To improve memory consumption and performance, Tarantool creates read views using the copy-on-write technique. In this case, duplication of the entire data set is not required: Tarantool duplicates only blocks modified after a read view is created.

**Примечание:** Tarantool Enterprise Edition supports read views starting from v2.11.0 and enables the ability to work with them using both *Lua* and *C API*.

### 8.6.1 Limitations

Read views have the following limitations:

- Only the *memtx* engine is supported.
- Only TREE and HASH *indexes* are supported.
- Pagination is not supported (the *after* option).

### 8.6.2 Working with read views

#### Creating a read view

To create a read view, call the *box.read_view.open()* function. The snippet below shows how to create a read view with the `read_view1` name.

```
tarantool> read_view1 = box.read_view.open({name = 'read_view1'})
```

After creating a read view, you can see the information about it by calling *read_view_object:info()*.

```
tarantool> read_view1:info()
---
- timestamp: 66.606817935
  signature: 24
  is_system: false
  status: open
  vclock: {1: 24}
  name: read_view1
  id: 1
...
```

To list all the created read views, call the *box.read_view.list()* function.

#### Querying data

After creating a read view, you can access database spaces using the *read_view_object.space* field. This field provides access to a space object that exposes the *select*, *get*, and *pairs* methods with the same behavior as corresponding `box.space` methods.

The example below shows how to select 4 records from the `bands` space:

```
tarantool> read_view1.space.bands:select({}, {limit = 4})
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
...
```

Similarly, you can retrieve data by the specific index.

```
tarantool> read_view1.space.bands.index.year:select({}, {limit = 4})
---
- - [4, 'The Beatles', 1960]
  - [2, 'Scorpions', 1965]
  - [1, 'Roxette', 1986]
  - [3, 'Ace of Base', 1987]
...
```

#### Closing a read view

When a read view is no longer needed, close it using the *read_view_object:close()* method because a read view may consume a substantial amount of memory.

```
tarantool> read_view1:close()
---
...
```

Otherwise, a read view is closed implicitly when the read view object is collected by the Lua garbage collector.

After the read view is closed, its *status* is set to `closed`. On an attempt to use it, an error is raised.

### 8.6.3 Example

A Tarantool session below demonstrates how to open a read view, get data from this view, and close it. To repeat these steps, you need to bootstrap a Tarantool instance as described in *Using data operations* (you can skip creating secondary indexes).

1. Insert test data.

   ```
   tarantool> bands:insert{1, 'Roxette', 1986}
              bands:insert{2, 'Scorpions', 1965}
              bands:insert{3, 'Ace of Base', 1987}
              bands:insert{4, 'The Beatles', 1960}
   ```

2. Create a read view by calling the `open` function. Then, make sure that the read view status is `open`.

   ```
   tarantool> read_view1 = box.read_view.open({name = 'read_view1'})

   tarantool> read_view1.status
   ---
   - open
   ...
   ```

3. Change data in a database using the `delete` and `update` operations.

   ```
   tarantool> bands:delete(4)
   ---
   - [4, 'The Beatles', 1960]
   ...
   tarantool> bands:update({2}, {{'=', 2, 'Pink Floyd'}})
   ---
   ```

(continues on next page)

(продолжение с предыдущей страницы)

```
- [2, 'Pink Floyd', 1965]
...
```

4. Query a read view to make sure it contains a snapshot of data before a database is updated.

```
tarantool> read_view1.space.bands:select()
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
...
```

5. Close a read view.

```
tarantool> read_view1:close()
---
...
```

### Read views: Lua API

This topic describes the Lua API for working with *read views*.

`box.read_view:open(`*{opts}*`)`
Create a *new read view*.

**Параметры**

- `opts` (*table*) – (optional) configurations options for a read view. For example, the `name` option specifies a read view name. If `name` is not specified, a read view name is set to `unknown`.

**Return** a created read view object

**Rtype** *read_view_object*

**Example:**

```
tarantool> read_view1 = box.read_view.open({name = 'read_view1'})
```

`object read_view_object`
An object that represents a *read view*.

`read_view_object:info()`
Get information about a read view such as a name, status, or ID. All the available fields are listed below in the object options.

**Return** information about a read view

**Rtype** *table*

`read_view_object:close()`
Close a read view. After the read view is closed, its *status* is set to `closed`. On an attempt to use it, an error is raised.

`read_view_object.status`
A read view status. The possible values are `open` and `closed`.

**Rtype** string

`read_view_object.id`
A unique numeric identifier of a read view.

**Rtype** number

`read_view_object.name`
A read view name. You can specify a read view name in the *box.read_view.open()* arguments.

**Rtype** string

`read_view_object.is_system`
Determine whether a read view is system. For example, system read views can be created to make a *checkpoint* or join a new *replica*.

**Rtype** boolean

`read_view_object.timestamp`
The *fiber.clock()* value at the moment of opening a read view.

**Rtype** number

`read_view_object.vclock`
The *box.info.vclock* value at the moment of opening a read view.

**Rtype** table

`read_view_object.signature`
The *box.info.signature* value at the moment of opening a read view.

**Rtype** number

`read_view_object.space`
Get access to database spaces included in a read view. You can use this field to *query space data*.

**Rtype** space object

### Read views: C API

This topic describes the C API for working with *read views*. The C API is MT-safe and provides the ability to use a read view from any thread, not only from the *main (TX) thread*.

The C API has the following specifics:

- The *space.upgrade* function is not applied to retrieved tuples even if a space upgrade is in progress.
- Tuples stored in *compressed spaces* are not decompressed - they are returned as *raw MessagePack* (`MP_EXT/MP_COMPRESSION`).

---

**Примечание:** You can learn how to call C code using stored procedures in the *C tutorial*.

---

### Data types

The opaque data types below represent raw read views and an iterator over data in a raw read view. Note that there is no special data type for tuples retrieved from a read view. Tuples are returned as raw MessagePack data (`const char *`).

typedef box_raw_read_view `box_raw_read_view_t`
: A raw database read view.

typedef box_raw_read_view_space `box_raw_read_view_space_t`
: A space in a raw read view.

typedef box_raw_read_view_index `box_raw_read_view_index_t`
: An index in a raw read view.

typedef box_raw_read_view_iterator `box_raw_read_view_iterator_t`
: An iterator over data in a raw read view.

### Creating and destroying read views

To create or destroy a read view, use the functions below.

*box_raw_read_view_t* *`box_raw_read_view_new`(const char *name)
: Open a raw read view with the specified name and get a pointer to this read view. In the case of error, returns `NULL` and sets *box_error_last()*. This function may be called from the main (TX) thread only.

  **Параметры**

  - `*name` (`const char`) – (optional) a read view name; if `name` is not specified, a read view name is set to `unknown`

  **Результат** a pointer to a read view

void `box_raw_read_view_delete`(*box_raw_read_view_t* *rv)
: Close a raw read view and release all resources associated with it. This function may be called from the main (TX) thread only.

  **Параметры**

  - `*rv` (*box_raw_read_view_t*) – a pointer to a read view

---

**Примечание:** Read views created using `box_raw_read_view_new` are displayed in *box.read_view.list()* along with read views *created in Lua*.

---

### Spaces and indexes

To fetch data from a read view, you need to specify an index to fetch the data from. The following functions are available for looking up spaces and indexes in a read view object.

*box_raw_read_view_space_t* *`box_raw_read_view_space_by_id`(const *box_raw_read_view_t* *rv, uint32_t space_id)
: Find a space by ID in a raw read view. If not found, returns `NULL` and sets *box_error_last()*.

  **Параметры**

  - `*rv` (`const` *box_raw_read_view_t*) – a pointer to a read view
  - `space_id` (`uint32_t`) – a space identifier

**Результат** a pointer to a space

*box_raw_read_view_space_t* *box_raw_read_view_space_by_name(const *box_raw_read_view_t* *rv, const char *space_name, uint32_t space_name_len)

Find a space by name in a raw read view. If not found, returns `NULL` and sets *box_error_last()*.

**Параметры**

- `*rv` (`const` *box_raw_read_view_t*) – a pointer to a read view
- `*space_name` (`const char`) – a space name
- `space_name_len` (`uint32_t`) – a space name length

**Результат** a pointer to a space

*box_raw_read_view_index_t* *box_raw_read_view_index_by_id(const *box_raw_read_view_space_t* *space, uint32_t index_id)

Find an index by ID in a read view's space. If not found, returns `NULL` and sets *box_error_last()*.

**Параметры**

- `*space` (`const` *box_raw_read_view_space_t*) – a pointer to a read view's space
- `space_id` (`uint32_t`) – a space identifier

**Результат** a pointer to an index

*box_raw_read_view_index_t* *box_raw_read_view_index_by_name(const *box_raw_read_view_space_t* *space, const char *index_name, uint32_t index_name_len)

Find an index by name in a read view's space. If not found, returns `NULL` and sets *box_error_last()*.

**Параметры**

- `*space` (`const` *box_raw_read_view_space_t*) – a pointer to a space
- `*index_name` (`const char`) – an index name
- `index_name_len` (`uint32_t`) – an index name length

**Результат** a pointer to an index

### Iteration and lookup

The functions below provide the ability to look up a tuple by the key or create an iterator over a read view index.

---

**Примечание:** Methods of the read view iterator are safe to call from any thread, but they may be used in one thread at the same time. This means that an iterator should be thread-local.

---

int box_raw_read_view_get(const *box_raw_read_view_index_t* *index, const char *key, const char *key_end, const char **data, uint32_t *size)

Look up a tuple in a read view's index. If found, the `data` and `size` out arguments return a pointer to and the size of tuple data. If not found, `*data` is set to `NULL` and `*size` is set to `0`.

**Параметры**

- `*index` (`const` *box_raw_read_view_index_t*) – a pointer to a read view's index

- `*key` (`const char`) – a pointer to the first byte of the MsgPack data that represents the search key
- `*key_end` (`const char`) – a pointer to the byte following the last byte of the MsgPack data that represents the search key
- `**data` (`const char`) – a pointer to the tuple data
- `*size` (`uint32_t`) – the size of tuple data

**Результат** `0` on success; in the case of error, returns `-1` and sets *box_error_last()*

int box_raw_read_view_iterator_create(*box_raw_read_view_iterator_t* *it, const *box_raw_read_view_index_t* *index, int type, const char *key, const char *key_end)

Create an iterator over a raw read view index. The initialized iterator object returned by this function remains valid and may be safely used until it's destroyed or the read view is closed. When the iterator object is no longer needed, it should be destroyed using *box_raw_read_view_iterator_destroy()*.

**Параметры**

- `*it` (*box_raw_read_view_iterator_t*) – an iterator over a raw read view index
- `*index` (`const` *box_raw_read_view_index_t*) – a pointer to a read view index
- `type` (`int`) – an iteration direction represented by the *iterator_type*
- `*key` (`const char`) – a pointer to the first byte of the MsgPack data that represents the search key
- `*key_end` (`const char`) – a pointer to the byte following the last byte of the MsgPack data that represents the search key

**Результат** `0` on success; in the case of error, returns `-1` and sets *box_error_last()*

int box_raw_read_view_iterator_next(*box_raw_read_view_iterator_t* *it, const char **data, uint32_t *size)

Retrieve the current tuple and advance the given iterator over a raw read view index. The pointer to and the size of tuple data are returned in the `data` and the `size` out arguments. The data returned by this function remains valid and may be safely used until the read view is closed.

**Параметры**

- `*it` (*box_raw_read_view_iterator_t*) – an iterator over a read view index
- `**data` (`const char`) – a pointer to the tuple data; at the end of iteration, `*data` is set to `NULL`
- `*size` (`uint32_t`) – the size of tuple data; at the end of iteration, `*size` is set to `0`

**Результат** `0` on success; in the case of error, returns `-1` and sets *box_error_last()*

void box_raw_read_view_iterator_destroy(*box_raw_read_view_iterator_t* *it)

Destroy an iterator over a raw read view index. The iterator object should not be used after calling this function, but the data returned by the iterator may be safely dereferenced until the read view is closed.

**Параметры**

- `*it` (*box_raw_read_view_iterator_t*) – an iterator over a read view index

### Space format

A space object's methods below provide the ability to get names and types of space fields.

uint32_t box_raw_read_view_space_field_count(const *box_raw_read_view_space_t* *space)
Get the number of fields defined in the format of a read view space.

**Параметры**

- *space (const *box_raw_read_view_space_t*) – a pointer to a read view space

**Результат** the number of fields

const char *box_raw_read_view_space_field_name(const *box_raw_read_view_space_t* *space, uint32_t field_no)
Get the name of a field defined in the format of a read view space. If the field number is greater than the total number of fields defined in the format, NULL is returned. The string returned by this function is guaranteed to remain valid until the read view is closed.

**Параметры**

- *space (const *box_raw_read_view_space_t*) – a pointer to a read view space
- field_no (uint32_t) – the field number (starts with 0)

**Результат** the name of a field

const char *box_raw_read_view_space_field_type(const *box_raw_read_view_space_t* *space, uint32_t field_no)
Get the type of a field defined in the format of a read view space. If the field number is greater than the total number of fields defined in the format, NULL is returned. The string returned by this function is guaranteed to remain valid until the read view is closed.

**Параметры**

- *space (const *box_raw_read_view_space_t*) – a pointer to a read view space
- field_no (uint32_t) – the field number (starts with 0)

**Результат** the type of a field

## 8.7 Flight recorder

**Example on GitHub**: flightrec

The flight recorder available in the Enterprise Edition is an event collection tool that gathers various information about a working Tarantool instance, such as:

- logs
- metrics
- requests and responses

This information helps you investigate incidents related to *crashing* a Tarantool instance.

### 8.7.1 Enable the flight recorder

The flight recorder is disabled by default and can be enabled and configured for a specific Tarantool instance. To enable the flight recorder, set the *flightrec.enabled* configuration option to `true`.

```
flightrec:
  enabled: true
```

After `flightrec.enabled` is set to `true`, the flight recorder starts collecting data in the flight recording file `current.ttfr`. This file is stored in the *snapshot.dir* directory. By default, the directory is `var/lib/{{ instance_name }}/<file_name>.ttfr`.

If the instance crashes and reboots, Tarantool rotates the flight recording: `current.ttfr` is renamed to `<timestamp>.ttfr` (for example, `20230411T050721.ttfr`) and the new `current.ttfr` file is created for collecting data. In the case of correct shutdown (for example, using `os.exit()`), Tarantool continues writing to the existing `current.ttfr` file after restart.

---

**Примечание:** Note that old flight recordings should be removed manually.

---

### 8.7.2 Configure a flight recorder

When the flight recorder is enabled, you can set the options related to *logging*, *metrics*, and storing the *request and response* data.

The `flightrec` configuration might look as follows:

```
flightrec:
  enabled: true
  logs_size: 10485800
  logs_log_level: 5
  metrics_period: 240
  metrics_interval: 0.5
  requests_size: 10485780
```

In the example, the following options are set:

- *flightrec.logs_size* – a log storage size in bytes.
- *flightrec.logs_log_level* – a *log_level*.
- *flightrec.metrics_period* – the number of seconds to store metrics after the dump.
- *flightrec.metrics_interval* – the frequency of metrics dumps in seconds.
- *flightrec.requests_size* – a storage size for the request and response data in bytes.

Read more: *Flight recorder configuration options*.

## 8.8 Журнал аудита

**Example on GitHub**: audit_log

The audit module available in Tarantool Enterprise Edition allows you to record various events occurred in Tarantool. Each *event* is an action related to authorization and authentication, data manipulation, administrator activity, or system events.

The module provides detailed reports of these activities and helps you find and fix breaches to protect your business. For example, you can see who created a new user and when.

It is up to each company to decide exactly what activities to audit and what actions to take. System administrators, security engineers, and people in charge of the company may want to audit different events for different reasons. Tarantool provides such an option for each of them.

### 8.8.1 Configure audit log

The section describes how to enable and configure audit logging and write logs to a selected destination – a file, a pipe, or a system logger.

Read more: *Audit log configuration reference*.

#### Enable audit logging

To enable audit logging, define the log location using the *audit_log.to* option in the configuration file. Possible log locations:

- a *file*
- a *pipe*
- a *system logger*

In the configuration below, the *audit_log.to* option is set to `file`. It means that the logs are written to a file. By default, audit logs are saved in the `var/log/{{ instance_name }}/audit.log` file. To specify the path to an audit log file explicitly, use the *audit_log.file* option.

```
audit_log:
  to: file
  file: 'audit_tarantool.log'
```

If you log to a file, Tarantool reopens the audit log at SIGHUP.

To disable audit logging, set the `audit_log.to` option to `devnull`.

#### Filter the events

Tarantool's extensive filtering options help you write only the events you need to the audit log. To select the recorded events, use the *audit_log.filter* option. Its value can be a list of events and event groups. You can customize the filters and use different combinations of them for your purposes. Possible filtering options:

- Filter by event. You can set a list of *events* to be recorded. For example, select `password_change` to monitor the users who have changed their passwords:

  ```
  audit_log:
    filter: [ password_change ]
  ```

- Filter by group. You can specify a list of *event groups* to be recorded. For example, select `auth` and `priv` to see the events related to authorization and granted privileges:

```
audit_log:
  filter: [ auth,priv ]
```

- Filter by group and event. You can specify a group and a certain event depending on the purpose. In the configuration below, `user_create`, `data_operations`, `ddl`, and `custom` are selected to see the events related to:
  - user creation
  - space creation, altering, and dropping
  - data modification or selection from spaces
  - *custom events* (any events added manually using the audit module API)

```
filter: [ user_create,data_operations,ddl,custom ]
```

### Set the format of audit log events

Use the *audit_log.format* option to choose the format of audit log events – plain text, CSV, or JSON.

```
format: json
```

JSON is used by default. It is more convenient to receive log events, analyze them, and integrate them with other systems if needed. The plain format can be efficiently compressed. The CSV format allows you to view audit log events in tabular form.

### Specify the spaces to be logged

The *audit_log.spaces* option is used to specify a list of space names for which data operation events should be logged.

In the configuration below, only the events from the `bands` space are logged:

```
spaces: [ bands ]
```

### Specify the logging mode in DML events

If set to `true`, the *audit_log.extract_key* option forces the audit subsystem to log the primary key instead of a full tuple in DML operations.

```
extract_key: true
```

### 8.8.2 Examples of audit log entries

In this example, the following audit log configuration is used:

```
audit_log:
  to: file
  file: 'audit_tarantool.log'
  filter: [ user_create,data_operations,ddl,custom ]
  format: json
  spaces: [ bands ]
  extract_key: true
```

Create a space `bands` and check the logs in the file after the creation:

```
box.schema.space.create('bands')
```

The audit log entry for the `space_create` event might look as follows:

```
{
  "time": "2024-01-24T11:43:21.566+0300",
  "uuid": "26af0a7d-1052-490a-9946-e19eacc822c9",
  "severity": "INFO",
  "remote": "unix/:(socket)",
  "session_type": "console",
  "module": "tarantool",
  "user": "admin",
  "type": "space_create",
  "tag": "",
  "description": "Create space Bands"
}
```

Then insert one tuple to space:

```
box.space.bands:insert { 1, 'Roxette', 1986 }
```

If the `extract_key` option is set to `true`, the audit system prints the primary key instead of the full tuple:

```
{
  "time": "2024-01-24T11:45:42.358+0300",
  "uuid": "b437934d-62a7-419a-8d59-e3b33c688d7a",
  "severity": "VERBOSE",
  "remote": "unix/:(socket)",
  "session_type": "console",
  "module": "tarantool",
  "user": "admin",
  "type": "space_insert",
  "tag": "",
  "description": "Insert key [2] into space bands"
}
```

If the `extract_key` option is set to `false`, the audit system prints the full tuple like this:

```
{
  "time": "2024-01-24T11:45:42.358+0300",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  "uuid": "b437934d-62a7-419a-8d59-e3b33c688d7a",
  "severity": "VERBOSE",
  "remote": "unix/:(socket)",
  "session_type": "console",
  "module": "tarantool",
  "user": "admin",
  "type": "space_insert",
  "tag": "",
  "description": "Insert tuple [1, \"Roxette\", 1986] into space bands"
}
```

### 8.8.3 Audit log events

#### Events types

The Tarantool audit log module can record various events that you can monitor and decide whether you need to take actions:

- Administrator activity – events related to actions performed by the administrator. For example, such logs record the creation of a user.
- Access events – events related to authorization and authentication of users. For example, such logs record failed attempts to access secure data.
- Data access and modification – events of data manipulation in the storage.
- System events – events related to modification or configuration of resources. For example, such logs record the replacement of a space.
- *Custom events* – any events added manually using the audit module API.

The full list of available audit log events is provided in the table below:

| Событие | Event type | Severity level | Пример |
|---|---|---|---|
| Audit log enabled for events | `audit_enable` | `VERBOSE` | |
| *Custom events* | `custom` | `INFO` (default) | |
| User authorized successfully | `auth_ok` | `VERBOSE` | `Authenticate user <USER>` |
| User authorization failed | `auth_fail` | `ALARM` | `Failed to authenticate user <USER>` |
| User logged out or quit the session | `disconnect` | `VERBOSE` | `Close connection` |
| User created | `user_create` | `INFO` | `Create user <USER>` |
| User dropped | `user_drop` | `INFO` | `Drop user <USER>` |
| Role created | `role_create` | `INFO` | `Create role <ROLE>` |
| Role dropped | `role_drop` | `INFO` | `Drop role <ROLE>` |
| User disabled | `user_disable` | `INFO` | `Disable user <USER>` |
| User enabled | `user_enable` | `INFO` | `Enable user <USER>` |
| User granted rights | `user_grant_rights` | `INFO` | `Grant <PRIVILEGE> rights for <OBJECT_TYPE> <OBJECT_NAME> to user <USER>` |
| User revoked rights | `user_revoke_rights` | `INFO` | `Revoke <PRIVILEGE> rights for <OBJECT_TYPE> <OBJECT_NAME> from user <USER>` |
| Role granted rights | `role_grant_rights` | `INFO` | `Grant <PRIVILEGE> rights for <OBJECT_TYPE> <OBJECT_NAME> to role <ROLE>` |
| Role revoked rights | `role_revoke_rights` | `INFO` | `Revoke <PRIVILEGE> rights for <OBJECT_TYPE> <OBJECT_NAME> from role <ROLE>` |
| User password changed | `password_change` | `INFO` | `Change password for user <USER>` |
| Failed attempt to access secure data (for example, personal records, details, geolocation) | `access_denied` | `ALARM` | `<ACCESS_TYPE> denied to <OBJECT_TYPE> <OBJECT_NAME>` |
| Expressions with arguments evaluated in a string | `eval` | `INFO` | `Evaluate expression <EXPR>` |
| Function called with arguments | `call` | `VERBOSE` | `Call function <FUNCTION> with arguments <ARGS>` |
| Iterator key selected from `space.index` | `space_select` | `VERBOSE` | `Select <ITER_TYPE> <KEY> from <SPACE>.<INDEX>` |
| Space created | `space_create` | `INFO` | `Create space <SPACE>` |
| Space altered | `space_alter` | `INFO` | `Alter space <SPACE>` |
| Space dropped | `space_drop` | `INFO` | `Drop space <SPACE>` |
| Tuple inserted into space | `space_insert` | `VERBOSE` | `Insert tuple <TUPLE> into space <SPACE>` |
| Tuple replaced in space | `space_replace` | `VERBOSE` | `Replace tuple <TUPLE> with <NEW_TUPLE> in space <SPACE>` |
| Tuple deleted from space | `space_delete` | `VERBOSE` | `Delete tuple <TUPLE> from space <SPACE>` |

**Примечание:** The `eval` event displays data from the `console` module and the `eval` function of the `net.box` module. For more on how they work, see *Module console* and *Module net.box – eval*. To separate the data, specify `console` or `binary` in the session field.

### Structure of audit log event

Each audit log event contains a number of fields that can be used to filter and aggregate the resulting logs. An example of a Tarantool audit log entry in JSON:

```
{
    "time": "2024-01-15T13:39:36.046+0300",
    "uuid": "cb44fb2b-5c1f-4c4b-8f93-1dd02a76cec0",
    "severity": "VERBOSE",
    "remote": "unix/:(socket)",
    "session_type": "console",
    "module": "tarantool",
    "user": "admin",
    "type": "auth_ok",
    "tag": "",
    "description": "Authenticate user Admin"
}
```

Each event consists of the following fields:

| Field | Описание | Пример |
|---|---|---|
| `time` | Time of the event | `2024-01-15T16:33:12.368+0300` |
| `uuid` | Since *3.0.0*. A unique identifier of audit log event | `cb44fb2b-5c1f-4c4b-8f93-1dd02a76cec0` |
| `severity` | Since *3.0.0*. A severity level. Each system audit event has a severity level determined by its importance. *Custom events* have the `INFO` severity level by default. | `VERBOSE` |
| `remote` | Remote host that triggered the event | `unix/:(socket)` |
| `session_type` | Session type | `console` |
| `module` | Audit log module. Set to `tarantool` for system events; can be overwritten for custom events | `tarantool` |
| `user` | User who triggered the event | `admin` |
| `type` | Audit event type | `auth_ok` |
| `tag` | A text field that can be overwritten by the user | |
| `description` | Human-readable event description | `Authenticate user Admin` |

#### Event groups

Built-in event groups are used to filter the event types that you want to audit. For example, you can set to record only authorization events or only events related to a space.

Tarantool provides the following event groups:

- `all` – all *events*.

  **Примечание:** Events `call` and `eval` are included only in the `all` group.

- `audit` – `audit_enable` event.
- `auth` – authorization events: `auth_ok`, `auth_fail`.
- `priv` – events related to authentication, authorization, users, and roles: `user_create`, `user_drop`, `role_create`, `role_drop`, `user_enable`, `user_disable`, `user_grant_rights`, `user_revoke_rights`, `role_grant_rights`, `role_revoke_rights`.
- `ddl` – events of space creation, altering, and dropping: `space_create`, `space_alter`, `space_drop`.
- `dml` – events of data modification in spaces: `space_insert`, `space_replace`, `space_delete`.
- `data_operations` – events of data modification or selection from spaces: `space_select`, `space_insert`, `space_replace`, `space_delete`.
- `compatibility` – events available in Tarantool before the version 2.10.0. `auth_ok`, `auth_fail`, `disconnect`, `user_create`, `user_drop`, `role_create`, `role_drop`, `user_enable`, `user_disable`, `user_grant_rights`, `user_revoke_rights`, `role_grant_rights`. `role_revoke_rights`, `password_change`, `access_denied`. This group enables the compatibility with earlier Tarantool versions.

**Предупреждение:** Be careful when recording `all` and `data_operations` event groups. The more events you record, the slower the requests are processed over time. It is recommended that you select only those groups whose events your company needs to monitor and analyze.

### 8.8.4 Custom events

Tarantool provides an API for writing custom audit log events. To enable these events, specify the `custom` value in the *audit_log.filter* option:

```
filter: [ user_create,data_operations,ddl,custom ]
```

#### Log a custom event

To log an event, use the `audit.log()` function that takes one of the following values:

- Message string. Printed to the audit log with type `message`:

  ```
  audit.log('Hello, Alice!')
  ```

- Format string and arguments. Passed to string format and then output to the audit log with type `message`:

```
audit.log('Hello, %s!', 'Bob')
```

- Table with audit log field values. The table must contain at least one field – `description`.

```
audit.log({ type = 'custom_hello', description = 'Hello, World!' })
audit.log({ type = 'custom_farewell', user = 'eve', module = 'custom', description␣
↪= 'Farewell, Eve!' })
```

Alternatively, you can use `audit.new()` to create a new log module. This allows you to avoid passing all custom audit log fields each time `audit.log()` is called. The `audit.new()` function takes a table of audit log field values (same as `audit.log()`). The `type` of the log module for writing custom events must either be `message` or have the `custom_` prefix.

```
local my_audit = audit.new({ type = 'custom_hello', module = 'my_module' })
my_audit:log('Hello, Alice!')
my_audit:log({ tag = 'admin', description = 'Hello, Bob!' })
```

### Overwrite custom event fields

It is possible to overwrite most of the custom audit log *fields* using `audit.new()` or `audit.log()`. The only audit log field that cannot be overwritten is `time`.

```
audit.log({ type = 'custom_hello', description = 'Hello!',
            session_type = 'my_session', remote = 'my_remote' })
```

If omitted, the `session_type` is set to the current session type, `remote` is set to the remote peer address.

---

**Примечание:** To avoid confusion with system events, the value of the type field must either be `message` (default) or begin with the `custom_` prefix. Otherwise, you receive the error message. Custom events are filtered out by default.

---

### Severity level

By default, custom events have the `INFO` *severity level*. To override the level, you can:

- specify the `severity` field
- use a shortcut function

The following shortcuts are available:

| Shortcut | Equivalent |
|---|---|
| `audit.verbose(...)` | `audit.log({severity = 'VERBOSE', ...})` |
| `audit.info(...)` | `audit.log({severity = 'INFO', ...})` |
| `audit.warning(...)` | `audit.log({severity = 'WARNING', ...})` |
| `audit.alarm(...)` | `audit.log({severity = 'ALARM', ...})` |

**Example**

```
audit.log({ severity = 'VERBOSE', description = 'Hello!' })
```

### 8.8.5 Tips

#### How many events can be recorded?

If you write to a file, the size of the Tarantool audit log is limited by the disk space. If you write to a system logger, the size of the Tarantool audit log is limited by the system logger. If you write to a pipe, the size of the Tarantool audit message is limited by the system buffer. If the `audit_log.nonblock = false`, if `audit_log.nonblock = true`, there is no limit.

#### How often should audit logs be reviewed?

Consider setting up a schedule in your company. It is recommended to review audit logs at least every 3 months.

#### How long should audit logs be stored?

It is recommended to store audit logs for at least one year.

#### What is the best way to process audit logs?

It is recommended to use SIEM systems for this issue.

## 8.9 Upgrading space schema

In Tarantool, *migration* refers to any change in a data schema, for example, creating an index, adding a field, or changing a field format. If you need to change a data schema, there are several possible cases:

- Schema migration does not require data migration: adding a field with the `is_nullable` parameter to the end of the space, creating an index.
- Schema migration requires data migration. For example, it is necessary when you have to iterate over the entire space to convert columns to a new format or remove the column completely.

To solve the task of migrating the data, you can:

- *Migrate data* to a new space manually.
- Use the `space:upgrade()` feature.

### 8.9.1 Space upgrade overview

The `space:upgrade()` feature allows users to upgrade the format of a space and the tuples stored in it without blocking the database.

### How to apply space upgrade

First, specify an upgrade function – a function that will convert the tuples in the space to a new format. The requirements for this function are listed below.

- The upgrade function takes two arguments. The first argument is a tuple to be upgraded. The second one is optional. It contains some additional information stored in plain Lua object. If omitted, the second argument is `nil`.
- The function returns a new tuple or a Lua table. For example, it can add a new field to the tuple. The new tuple must conform to the new space format set by the upgrade operation.
- The function should be registered with *box.schema.func.create*. It should also be stored, deterministic, and written in Lua.
- The function should not change the primary key of the tuple.
- The function should be idempotent: `f(f(t)) = f(t)`. This is necessary because the function is applied to all tuples returned to the user, and some of them may have already been upgraded in the background.

Then define a new space format. This step is optional. However, it could be useful if, for example, you want to add a new column with data. For details, check the *Usage Example* section.

The next optional step is to choose an upgrade mode. There are three modes: `upgrade`, `dryrun`, and `dryrun+upgrade`. The default value is `upgrade`. To check an upgrade function without applying any changes, choose the `dryrun` mode. To run a space upgrade without testing the function, pick the `upgrade` mode. If you want to apply both the test and the actual upgrade, use the `dryrun+upgrade` option. For details, see the *Upgrade Modes* section.

### How the upgrade works

The user defines an upgrade function. Each tuple of the chosen space is passed through the function. The function converts the tuple from the old format to a new one. The function is applied to all tuples stored in the space in the background. Besides, the function is applied to all tuples returned to the user via the box API (for example, `select`, `get`). Therefore, it appears that the space upgrades instantly.

Keep in mind that `space:upgrade` differs from the *space_object:format()* in the following ways:

| Difference | `space:upgrade()` | `space:format()` |
|---|---|---|
| Non-blocking | Yes. It returns tuples in the new format, whether or not they have already been converted. | Yes. |
| Set a format incompatible with the current one | Yes. Works for non-indexed field types only. | No, only expand the format in a compatible way. |
| Visibility of changes | Immediately. All changes are visible and replicated immediately. New data should conform to the new format immediately after the call. | After data validation. Data validation starts in the background, it does not block the database. Inserting data incompatible with the new format is allowed before validation is completed – in this case `space.format` fails. |
| Cancel (error/restart) | Writes the state to the system table. Restart: the operation continues. Error: the operation should be restarted manually, any other attempt to change the table fails. | Leaves no traces. |
| Set the upgrade function | Yes. The upgrade may take a while to traverse the space and transform tuples. | No. |

**Примечание:** At the moment, the feature is not supported for *vinyl* spaces.

### 8.9.2 User API

The `space:upgrade()` method is added to the *space object*:

`space:upgrade(`*{func*[, *arg*, *format*, *mode*, *is_ async]}*]`)`

**Параметры**

- `func` (`string/integer`) – upgrade function name (string) or ID (integer). For details, see the *upgrade function requirements* section.
- `arg` – additional information passed to the upgrade function in the second argument. The option accepts any Lua value that can be encoded in MsgPack, which means that the *msgpack.encode(arg)* should succeed. For example, one can pass a scalar or a Lua table. The default value is `nil`.
- `format` (`map`) – new space format. The requirements for this are the same as for any other *space:format()*. If the field is omitted, the space format will remain the same as before the upgrade.
- `mode` (*`string`*) – *upgrade mode*. Possible values: `upgrade`, `dryrun`, `dryrun+upgrade`. The default value is `upgrade`.
- `is_async` (`boolean`) – the flag indicates whether to wait until the upgrade operation is complete before exiting the function. The default value is `false` – the function is blocked until the upgrade operation is finished.

**Return** object describing the status of the operation (also known as `future`). The methods of the object are described below.

`object future_object`

`future_object:info`(*dryrun*, *status*, *func*, *arg*, *owner*, *error*, *progress*)
Shows information about the state of the upgrade operation.

**Параметры**

- `dryrun` (`boolean`) – dry run mode flag. Possible values: `true` for a dry run, `nil` for an actual upgrade.
- `status` (*string*) – *upgrade status*. Possible values: `inprogress`, `waitrw`, `error`, `replica`, `done`.
- `func` (`string/integer`) – name of the upgrade function. It is the same as passed to the `space:upgrade` method. The field is `nil` if the `status` is `done`.
- `arg` – additional information passed to the upgrade function. It is the same as for the `space:upgrade` method. The field is `nil` if it is omitted in the `space:upgrade`.
- `owner` (*string*) – UUID of the instance running the upgrade (see *box.info.uuid*). The field is `nil` if the `status` is `done`.
- `error` (*string*) – error message if the `status` is `error`, otherwise `nil`.
- `progress` (*string*) – completion percentage if the `status` is `inprogress`/`waitrw`, otherwise `nil`.

**Return** a table with information about the state of the upgrade operation

**Rtype** *table*

The fields can also be accessed directly, without calling the `info()` method. For example, `future.status` is the same as `future:info().status`.

`future_object:wait`([*timeout*])
Waits until the upgrade operation is completed or a timeout occurs. An operation is considered completed if its status is `done` or `error`.

**Параметры**

- `timeout` (`double`) – if the `timeout` argument is omitted, the method waits as long as it takes.

**Return** returns `true` if the operation has been completed, `false` on timeout

**Rtype** boolean

`future_object:cancel()`
Cancels the upgrade operation if it is currently running. Otherwise, an exception is thrown. A canceled upgrade operation completes with an error.

**Return** none

**Rtype** void

Running `space:upgrade()` with `is_async = false` or the `is_async` field not set is equal to:

```
local future = space:upgrade({func = 'my_func', is_async = true})
future:wait()
return future
```

If called without arguments, `space:upgrade()` returns a future object for the active upgrade operation. If there is none, it returns `nil`.

### 8.9.3 Upgrade modes

There are three upgrade modes: `dryrun`, `dryrun+upgrade`, and `upgrade`. Regardless of the mode selected, the upgrade does not block execution. Once in a while, the background fiber commits the upgraded tuples and yields.

Calling `space:upgrade` without arguments always returns the current state of the space upgrade, never the state of a dry run. If there is a dry run working in the background, `space:upgrade` will still return `nil`. Unlike an actual space upgrade, the future object returned by a dry run upgrade can't be recovered if it is lost. So a dry run is aborted if it is garbage collected.

**Предупреждение:** In `dryrun+upgrade` mode: if the future object is garbage collected by Lua before the end of the dry run and the start of the upgrade, then the dry run will be canceled, and no upgrade will be started.

Upgrade modes:

- `upgrade` mode: the background fiber iterates over the space, applies the upgrade function, checks that obtained tuples fit the new space format, and updates the tuples. This mode prevents the space from being altered. The mode can only be performed on the master instance.
- `dryrun` mode: the dry-run mode is used to check the upgrade function. The mode does not apply any changes to the target space. It starts a background fiber. The fiber:
  - Iterates over the target space.
  - Attempts to apply the upgrade function to each tuple stored in the space.
  - Checks if the returned tuple matches the new format.
  - Checks if the function is idempotent.
  - Checks that the function does not modify the primary key.

  For details, see the *upgrade function requirements* section.

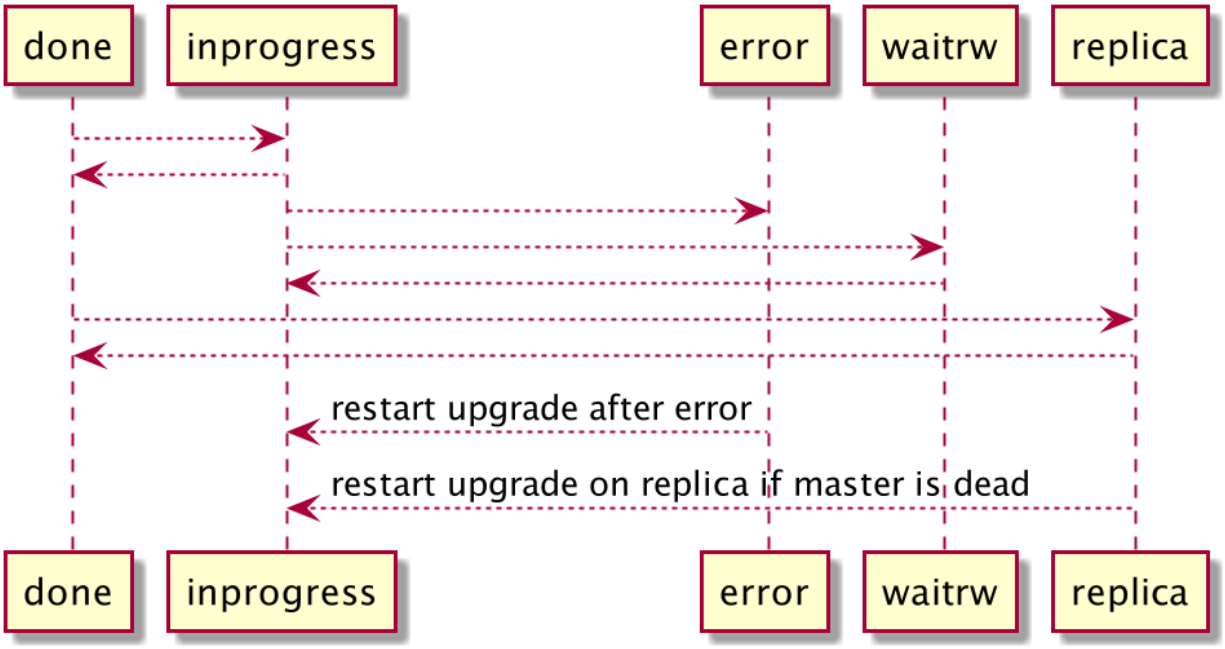
  To start a dry run, pass `mode='dryrun'` to the `space:upgrade` method. In this case, the future object has the `dryrun` field set to `true`. The possible statuses are `inprogress` and `dryrun`. `replica` and `waitrw` states are never set for a dry run future object.

  The `dryrun` mode is not persisted. Restarting the instance does not restart a dry run. A dry run only works on the original instance, never on replicas. Unlike a real upgrade, a dry run does not prevent the space from being altered. The space can even be dropped. In this case, the dry run will complete with an error.
- `dryrun+upgrade` mode: it starts a dry run, which, if completed successfully, triggers an actual upgrade. The future object returned by `space:upgrade` remains valid throughout the process. It starts as the future object of the dry run. Then, under the hood, it is converted into an upgrade future object. Waiting on it would wait for both the dry run and the upgrade to complete. During the dry run, the future object has the `dryrun` field set to `true`. When the actual upgrade starts, the `dryrun` field is set to `nil`. The mode can only be performed on the master instance.

### 8.9.4 States

An upgrade operation has one of the following upgrade states:

- `inprogress` – the upgrade operation is running in the background. The function is applied to all tuples returned to the user.
- `waitrw` – the instance was switched to the read-only mode (for example, by using `box.cfg.read_only`), so the upgrade couldn't proceed. The upgrade process will resume as soon as the instance switches back to read-write mode. Nevertheless, the upgrade function is applied to all tuples returned to the user.
- `error` – the upgrade operation failed with an error. See the `error` field for the error message. See the log for the tuple that caused the error. No alter operation is allowed, except for another upgrade, supposed to fix the problem. Nevertheless, the upgrade function is applied to all tuples returned to the user. The space is writable.
- `done` – the upgrade operation is successfully completed. The upgrade function is not applied to tuples returned to the user anymore. The function can be deleted.
- `replica` – the upgrade operation is either running or completed with an error on another instance. See the `owner` field for the UUID of the instance running the upgrade. Nevertheless, the upgrade function is applied to all tuples returned to the user.




### 8.9.5 Interaction with alter

While a space upgrade is in progress, the space can't be altered or dropped. The attempt to do that will throw an exception. Restarting an upgrade is allowed in case the currently running upgrade is canceled or completed with an error. It means the manual restart is possible if the upgrade operation is in the *error state*.

If a space upgrade was canceled or failed with an error, the space can't be altered or dropped. The only option is to restart the upgrade using a different upgrade function or format.

### 8.9.6 Interaction with recovery

The space upgrade state is persisted. It is stored in the `_space` system table. If an instance with a space upgrade in progress (`inprogress` state) is shut down, it restarts the space upgrade after recovery. If a space upgrade fails (switches to the `error` state), it remains in the error state after recovery.

### 8.9.7 Interaction with replication

The changes made to a space by a space upgrade are replicated. Just as on the instance where the upgrade is performed, the upgrade function is applied to all tuples returned to the user on the replicas. However, the upgrade operation is not performed on the replicas in the background. The replicas wait for the upgrade operation to complete on the master. They can't alter or drop the space. Normally, they can't cancel or restart the upgrade operation either.

There is an emergency exception when the master is permanently dead. It is possible to restart a space upgrade that started on another instance. The restart is possible if the upgrade owner UUID (see the `owner` field) has been deleted from the `_cluster` system table.

---

**Примечание:** Except the `dryrun` mode, the upgrade can only be performed on the master. If the instance is no longer the master, the upgrade is suspended until the instance is master again. Restarting the upgrade on a new master works only if the old one has been removed from the replica set (`_cluster` system space).

---

### 8.9.8 Usage example

Suppose there are two columns in the space `test` – `id` (unsigned) and `data` (string). The example shows how to upgrade the schema and add another column to the space using `space:upgrade()`. The new column contains the `id` values converted to string. Each step takes a while.

The test space is generated with the following script:

```
local log = require('log')
box.cfg{
    checkpoint_count = 1,
    memtx_memory = 5 * 1024 * 1024 * 1024,
}
box.schema.space.create('test')
box.space.test:format{
    {name = 'id', type = 'unsigned'},
    {name = 'data', type = 'string'},
}
box.space.test:create_index('pk')
local count = 20 * 1000 * 1000
local progress = 0
box.begin()
for i = 1, count do
    box.space.test:insert{i, 'data' .. i}

    if i % 1000 == 0 then
        box.commit()
        local p = math.floor(i / count * 100)
        if progress ~= p then
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            progress = p
            log.info('Generating test data set... %d%% done', p)
        end
        box.begin()
    end
end
box.commit()
box.snapshot()
os.exit(0)
```

To upgrade the space, connect to the server and then run the commands below:

```
localhost:3301> box.schema.func.create('convert', {
              >     language = 'lua',
              >     is_deterministic = true,
              >     body = [[function(t)
              >         if #t == 2 then
              >             return t:update({{'!', 2, tostring(t.id)}})
              >         else
              >             return t
              >         end
              >     end]],
              > })
localhost:3301> box.space.test:upgrade({
              >     func = 'convert',
              >     format = {
              >         {name = 'id', type = 'unsigned'},
              >         {name = 'id_string', type = 'string'},
              >         {name = 'data', type = 'string'},
              >     },
              > })
```

While the upgrade is in progress, you can track the state of the upgrade. To check the status, connect to Tarantool from another console and run the following commands:

```
localhost:3311> box.space.test:upgrade()
---
- status: inprogress
  progress: 8%
  owner: 579a9e99-427e-4e99-9e2e-216bbd3098a7
  func: convert
...
```

Even though the upgrade is only 8% complete, selecting the data from the space returns the converted tuples:

```
localhost:3311> box.space.test:select({}, {iterator = 'req', limit = 5})
---
- - [20000000, '20000000', 'data20000000']
  - [19999999, '19999999', 'data19999999']
  - [19999998, '19999998', 'data19999998']
  - [19999997, '19999997', 'data19999997']
  - [19999996, '19999996', 'data19999996']
...
```

**Примечание:** The tuples contain the new field even though the space upgrade is still running.

Wait for the space upgrade to complete using the command below:

```
localhost:3311> box.space.test:upgrade():wait()
```

## 8.10 Метрики системы мониторинга

| Параметр | Description | Тип по SNMP | Единицы измерения | Предел |
|---|---|---|---|---|
| Version | Версия Tarantool | DisplayString | | |
| IsAlive | Показатель доступности экземпляра | Integer (список) | | 0 - недоступен<br>1 - доступен |
| MemoryLua | объем памяти, занятый Lua | Gauge32 | МБайт | 900 |
| MemoryData | объем, используемый для хранения данных | Gauge32 | МБайт | значение задается вручную |
| MemoryNet | объем, используемый для сетевого ввода-выводы | Gauge32 | МБайт | 1024 |
| MemoryIndex | объем, используемый для хранения индексов | Gauge32 | МБайт | значение задается вручную |
| MemoryCache | объем, используемый для хранения кэша (только для vinyl) | Gauge32 | МБайт | |
| ReplicationLag | время задержки с момента последней синхронизации (максимальное значение, если есть несколько файберов) | Integer32 | сек. | 5 |
| FiberCount | количество файберов | Gauge32 | шт. | 1000 |
| CurrentTime | текущее время в секундах, с 1 января 1970г. | Unsigned32 | временная отметка Unix в сек. | |
| StorageStatus | статус набора реплик | Integer | список | $> 1$ |
| StorageAlerts | количество | Gauge32 | шт. | $>= 1$ |
| StorageTotalBkts | общее количество сегментов в хранилище | Gauge32 | шт. | $< 0$ |

continues on next page

Таблица 1 – продолжение с предыдущей страницы

| Параметр | Description | Тип по SNMP | Единицы измерения | Предел |
|---|---|---|---|---|
| StorageActiveBkts | количество сегментов в статусе ACTIVE | Gauge32 | шт. | < 0 |
| StorageGarbageBkts | количество сегментов в статусе GARBAGE | Gauge32 | шт. | < 0 |
| StorageReceivingBkts | количество сегментов в статусе RECEIVING | Gauge32 | шт. | < 0 |
| StorageSendingBkts | количество сегментов в статусе SENDING | Gauge32 | шт. | < 0 |
| RouterStatus | статус роутера | Integer | список | > 1 |
| RouterAlerts | количество предупреждений для роутера | Gauge32 | шт. | >= 1 |
| RouterKnownBkts | количество сегментов в известных наборах реплик | Gauge32 | шт. | < 0 |
| RouterUnknownBkts | количество реплик, неизвестных роутеру | Gauge32 | шт. | < 0 |
| RequestCount | общее количество запросов | Counter64 | шт. | |
| InsertCount | общее количество запросов вставки | Counter64 | шт. | |
| DeleteCount | общее количество запросов на удаление | Counter64 | шт. | |
| ReplaceCount | общее количество запросов замены | Counter64 | шт. | |
| UpdateCount | общее количество запросов на обновление | Counter64 | шт. | |
| SelectCount | общее количество запросов выборки | Counter64 | шт. | |
| EvalCount | количество вызовов через Eval | Counter64 | шт. | |
| CallCount | количество вызовов через `call` | Counter64 | шт. | |
| ErrorCount | количество ошибок в Tarantool | Counter64 | шт. | |
| AuthCount | количество завершенных операций по аутентификации | Counter64 | шт. | |

## 8.11 Устаревшие функции

Прекращена поддержка ZooKeeper и `orchestrator`. Тем не менее, при необходимости их можно использовать.

В следующих разделах описаны соответствующие функциональные возможности.

### 8.11.1 Управление кластером по API

Для управления кластером используйте оркестратор `orchestrator`, включенный в комплект поставки. `orchestrator` использует ZooKeeper для хранения и передачи конфигурации. Для управления кластером в `orchestrator` есть REST API. Изменение конфигураций в ZooKeeper осуществляется в результате вызова API-функций из `orchestrator`, что, в свою очередь, приводит к изменениям конфигурации узлов Tarantool.

Мы рекомендуем использовать интерфейс командной строки **curl** для вызова API-функций из `orchestrator`.

В следующем примере показано, как зарегистрировать новую зону доступности (DC):

```
$ curl -X POST http://HOST:PORT/api/v1/zone \
    -d '{
  "name": "Caucasian Boulevard"
  }'
```

Чтобы проверить, была ли выполнена регистрация DC, попробуйте использовать следующую команду. Результатом будет список всех зарегистрированных узлов в формате JSON:

```
$ curl http://HOST:PORT/api/v1/zone| python -m json.tool
```

Чтобы применить новую конфигурацию непосредственно к узлам Tarantool, увеличьте номер версии конфигурации после вызова API-функции. Для этого используйте запрос POST к `/api/v1/version`:

```
$ curl -X POST http://HOST:PORT/api/v1/version
```

В целом, чтобы обновить конфигурацию кластера:

1. Call the `POST/PUT` method of the `orchestrator`. As a result, the ZooKeeper nodes are updated, and a subsequent update of the Tarantool nodes is initiated.
2. Update the configuration version using the `POST` request to `/api/v1/version`. As a result, the configuration is applied to the Tarantool nodes.

Для получения дополнительной информации об API оркестратора см. *Справочник по Orchestrator API*.

### 8.11.2 Настройка геодублирования

По логике узлы кластера могут относиться к некоторой зоне доступности. Физически же зона доступности – это отдельный DC или стойка внутри DC. Можно задать матрицу весов (расстояний) для зон доступности.

Новые зоны добавляются путем вызова соответствующего метода API оркестратора.

По умолчанию матрица весов (расстояний) для зон не настроена, а геодублирование для таких конфигураций работает следующим образом:

- Данные всегда записываются на мастер.

- Если мастер доступен, то он используется для чтения.
- Если мастер недоступен, то любая доступная реплика используется для чтения.

Когда вы задаете матрицу весов (расстояний), вызывая `/api/v1/zones/weights`, система автоматического масштабирования СУБД Tarantool находит реплику, которая ближе всех к указанному роутеру по весам, и начинает использовать эту реплику для чтения. Если эта реплика недоступна, то выбирается следующая ближайшая реплика с учетом расстояний, указанных в конфигурации.

### Справочник по Orchestrator API

### Configuring the zones

- *POST /api/v1/zone*
- *GET /api/v1/zone/*
- *PUT /api/v1/zone/*
- *DELETE /api/v1/zone/*

`POST /api/v1/zone`
Создание новой зоны.

**Запрос**

```
{
"name": "zone 1"
}
```

**Ответ**

```
{
"error": {
    "code": 0,
    "message": "ok"
},
"data": {
    "id": 2,
    "name": "zone 2"
},
"status": true
}
```

**Возможные ошибки**

- `zone_exists` – указанная зона уже существует

`GET /api/v1/zone/{zone_id: optional}`
Возврат информации об указанной зоне или обо всех зонах.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "data": [
        {
            "id": 1,
            "name": "zone 11"
        },
        {
            "id": 2,
            "name": "zone 2"
        }
    ],
    "status": true
}
```

**Возможные ошибки**

- `zone_not_found` – указанная зона не обнаружена

`PUT /api/v1/zone/{zone_id}`

Обновление информации о зоне.

**Тело**

```
{
    "name": "zone 22"
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `zone_not_found` – указанная зона не обнаружена

`DELETE /api/v1/zone/{zone_id}`

Удаление зоны, если в ней нет узлов.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `zone_not_found` – указанная зона не обнаружена
- `zone_in_use` – в указанной зоне есть хотя бы один узел

### Настройка веса зоны

- *GET /api/v1/zones/weights*
- *POST /api/v1/zones/weights*

`POST /api/v1/zones/weights`
Конфигурация веса зоны.

**Тело**

```
{
    "weights": {
        "1": {
            "2": 10,
            "3": 11
        },
        "2": {
            "1": 10,
            "3": 12
        },
        "3": {
            "1": 11,
            "2": 12
        }
    }
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `zones_weights_error` – ошибка конфигурации

`GET /api/v1/zones/weights`
Возврат конфигурации веса зоны.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
```

(continues on next page)

(продолжение с предыдущей страницы)


```
    },
    "data": {
        "1": {
            "2": 10,
            "3": 11
        },
        "2": {
            "1": 10,
            "3": 12
        },
        "3": {
            "1": 11,
            "2": 12
        }
    },
    "status": true
}
```


**Возможные ошибки**

- `zone_not_found` – указанная зона не обнаружена

### Настройка реестра

- *GET /api/v1/registry/nodes/new*
- *POST /api/v1/registry/node*
- *PUT /api/v1/registry/node/*
- *GET /api/v1/registry/node/*
- *DELETE /api/v1/registry/node/*

`GET /api/v1/registry/nodes/new`

Возврат всех обнаруженных узлов.

**Ответ**


```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": [
        {
            "uuid": "uuid-2",
            "hostname": "tnt2.public.i",
            "name": "tnt2"
        }
    ],
    "status": true
}
```

`POST /api/v1/registry/node`
Регистрация обнаруженного узла.

**Тело**

```
{
    "zone_id": 1,
    "uuid": "uuid-2",
    "uri": "tnt2.public.i:3301",
    "user": "user1:pass1",
    "repl_user": "repl_user1:repl_pass1",
    "cfg": {
        "listen": "0.0.0.0:3301"
    }
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `node_already_registered` – указанный узел уже зарегистрирован
- `zone_not_found` – указанная зона не обнаружена
- `node_not_discovered` – указанный узел не обнаружен

`PUT /api/v1/registry/node/{node_uuid}`
Обновление параметров зарегистрированного узла.

**Тело**

Передача только обновляемых параметров.

```
{
    "zone_id": 1,
    "repl_user": "repl_user2:repl_pass2",
    "cfg": {
        "listen": "0.0.0.0:3301",
        "memtx_memory": 100000
    }
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
```

(continues on next page)

(продолжение с предыдущей страницы)


```
    },
    "data": {},
    "status": true
}
```


**Возможные ошибки**

- `node_not_registered` – указанный узел не зарегистрирован

`GET /api/v1/registry/node/{node_uuid: optional}`

Возврат информации об узлах в кластере. Если передать `node_uuid`, вернется информация только по данному узлу.

**Ответ**


```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {
        "uuid-1": {
            "user": "user1:pass1",
            "hostname": "tnt1.public.i",
            "repl_user": "repl_user2:repl_pass2",
            "uri": "tnt1.public.i:3301",
            "zone_id": 1,
            "name": "tnt1",
            "cfg": {
                "listen": "0.0.0.0:3301",
                "memtx_memory": 100000
            },
            "zone": 1
        },
        "uuid-2": {
            "user": "user1:pass1",
            "hostname": "tnt2.public.i",
            "name": "tnt2",
            "uri": "tnt2.public.i:3301",
            "repl_user": "repl_user1:repl_pass1",
            "cfg": {
                "listen": "0.0.0.0:3301"
            },
            "zone": 1
        }
    },
    "status": true
}
```


**Возможные ошибки**

- `node_not_registered` – указанный узел не зарегистрирован

`DELETE /api/v1/registry/node/{node_uuid}`

Удаление узла, если он не относится ни к одному набору реплик.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `node_not_registered` – указанный узел не зарегистрирован
- `node_in_use` – указанный узел используется в наборе реплик

### API по роутерам

- *GET /api/v1/routers*
- *POST /api/v1/routers*
- *DELETE /api/v1/routers/{uuid}*

`GET /api/v1/routers`

Возврат списка всех узлов, которые представляют собой роутер.

**Ответ**

```
{
    "data": [
        "uuid-1"
    ],
    "status": true,
    "error": {
        "code": 0,
        "message": "ok"
    }
}
```

`POST /api/v1/routers`

Присвоение узлу роли роутера.

**Тело**

```
{
    "uuid": "uuid-1"
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `node_not_registered` – указанный узел не зарегистрирован

`DELETE /api/v1/routers/{uuid}`

Снятие роли роутера с узла.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

### Настройка наборов реплик

- *POST /api/v1/replicaset*
- *PUT /api/v1/replicaset/*
- *GET /api/v1/replicaset/*
- *DELETE /api/v1/replicaset/*
- *POST /api/v1/replicaset/{replicaset_uuid}/master*
- *POST /api/v1/replicaset/{replicaset_uuid}/node*
- *DELETE /api/v1/zone/*

`POST /api/v1/replicaset`

Создание набора реплик со всеми зарегистрированными узлами.

**Тело**

```
{
    "uuid": "optional-uuid",
    "replicaset": [
        {
            "uuid": "uuid-1",
            "master": true
        }
    ]
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {
        "replicaset_uuid": "cc6568a2-63ca-413d-8e39-704b20adb7ae"
    },
    "status": true
}
```

**Возможные ошибки**

- `replicaset_exists` – указанный набор реплик уже существует
- `replicaset_empty` – указанный набор реплик не содержит ни одного узла
- `node_not_registered` – указанный узел не зарегистрирован
- `node_in_use` – указанный узел используется в другом наборе реплик

`PUT /api/v1/replicaset/{replicaset_uuid}`

Обновление параметров набора реплик.

**Тело**

```
{
    "replicaset": [
        {
            "uuid": "uuid-1",
            "master": true
        },
        {
            "uuid": "uuid-2",
            "master": false,
            "off": true
        }
    ]
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `replicaset_empty` – указанный набор реплик не содержит ни одного узла
- `replicaset_not_found` – указанный набор реплик не обнаружен
- `node_not_registered` – указанный узел не зарегистрирован

- `node_in_use` – указанный узел используется в другом наборе реплик

`GET /api/v1/replicaset/{replicaset_uuid: optional}`

Возврат информации обо всех компонентах кластера. Если передать `replicaset_uuid`, вернется информация только по данному набору реплик.

**Тело**

```
{
    "name": "zone 22"
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {
        "cc6568a2-63ca-413d-8e39-704b20adb7ae": {
            "uuid-1": {
                "hostname": "tnt1.public.i",
                "off": false,
                "repl_user": "repl_user2:repl_pass2",
                "uri": "tnt1.public.i:3301",
                "master": true,
                "name": "tnt1",
                "user": "user1:pass1",
                "zone_id": 1,
                "zone": 1
            },
            "uuid-2": {
                "hostname": "tnt2.public.i",
                "off": true,
                "repl_user": "repl_user1:repl_pass1",
                "uri": "tnt2.public.i:3301",
                "master": false,
                "name": "tnt2",
                "user": "user1:pass1",
                "zone": 1
            }
        }
    },
    "status": true
}
```

**Возможные ошибки**

- `replicaset_not_found` – указанный набор реплик не обнаружен

`DELETE /api/v1/replicaset/{replicaset_uuid}`

Удаление набора реплик.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `replicaset_not_found` – указанный набор реплик не обнаружен

`POST /api/v1/replicaset/{replicaset_uuid}/master`
Смена мастера в наборе реплик.

**Тело**

```
{
    "instance_uuid": "uuid-1",
    "hostname_name": "hostname:instance_name"
}
```

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Возможные ошибки**

- `replicaset_not_found` – указанный набор реплик не обнаружен
- `node_not_registered` – указанный узел не зарегистрирован
- `node_not_in_replicaset` – указанный узел находится не в указанном наборе реплик

`POST /api/v1/replicaset/{replicaset_uuid}/node`
Добавление узла в набор реплик.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

**Тело**

```
{
    "instance_uuid": "uuid-1",
    "hostname_name": "hostname:instance_name",
    "master": false,
    "off": false
}
```

**Возможные ошибки**

- `replicaset_not_found` – указанный набор реплик не обнаружен
- `node_not_registered` – указанный узел не зарегистрирован
- `node_in_use` – указанный узел используется в другом наборе реплик

`GET /api/v1/replicaset/status`

Возврат статистики по кластеру.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {
        "cluster": {
            "routers": [
                {
                    "zone": 1,
                    "name": "tnt1",
                    "repl_user": "repl_user1:repl_pass1",
                    "hostname": "tnt1.public.i",
                    "status": null,
                    "uri": "tnt1.public.i:3301",
                    "user": "user1:pass1",
                    "uuid": "uuid-1",
                    "total_rps": null
                }
            ],
            "storages": [
                {
                    "hostname": "tnt1.public.i",
                    "repl_user": "repl_user2:repl_pass2",
                    "uri": "tnt1.public.i:3301",
                    "name": "tnt1",
                    "total_rps": null,
                    "status": 'online',
                    "replicas": [
                        {
                            "user": "user1:pass1",
                            "hostname": "tnt2.public.i",
                            "replication_info": null,
                            "repl_user": "repl_user1:repl_pass1",
                            "uri": "tnt2.public.i:3301",
```

(continues on next page)

(продолжение с предыдущей страницы)


```
                            "uuid": "uuid-2",
                            "status": 'online',
                            "name": "tnt2",
                            "total_rps": null,
                            "zone": 1
                        }
                    ],
                    "user": "user1:pass1",
                    "zone_id": 1,
                    "uuid": "uuid-1",
                    "replicaset_uuid": "cc6568a2-63ca-413d-8e39-704b20adb7ae",
                    "zone": 1
                }
            ]
        }
    },
    "status": true
}
```


**Возможные ошибки**

- `zone_not_found` – указанная зона не обнаружена
- `zone_in_use` – в указанной зоне есть хотя бы один узел

### Настройка версии конфигурации

- *POST /api/v1/version*
- *GET /api/v1/version*

`POST /api/v1/version`
Настройка версии конфигурации.

**Ответ**


```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "status": true,
    "data": {
        "version": 2
    }
}
```


**Возможные ошибки**

- `cfg_error` – ошибка конфигурации

`GET /api/v1/version`
Возврат версии конфигурации.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "status": true,
    "data": {
        "version": 2
    }
}
```

### Конфигурация шардинга

- *POST /api/v1/sharding/cfg*
- *GET /api/v1/sharding/cfg*

`POST /api/v1/sharding/cfg`
Добавление новой конфигурации шардинга.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "status": true,
    "data": {}
}
```

`GET /api/v1/sharding/cfg`
Возврат текущей конфигурации шардинга.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "status": true,
    "data": {}
}
```

#### Сброс конфигурации кластера

- *POST /api/v1/clean/cfg*
- *POST /api/v1/clean/all*

`POST /api/v1/clean/cfg`
Сброс конфигурации кластера.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "status": true,
    "data": {}
}
```

`POST /api/v1/clean/all`
Сброс конфигурации кластера и удаление информации об узлах кластера из каталогов ZooKeeper.

**Ответ**

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "status": true,
    "data": {}
}
```

## 8.12 Modules

В данной главе рассматриваются Lua-модули с открытым и закрытым исходным кодом для Tarantool Enterprise, которые включены в дистрибутив в качестве автономного репозитория сторонних библиотек.

### 8.12.1 Модули с открытым исходным кодом:

- avro-schema – набор инструментов для схемы Apache Avro;
- *checks* is a type checker of functional arguments. This library that declares a `checks()` function and `checkers` table that allow to check the parameters passed to a Lua function in a fast and unobtrusive way.
- http – это встроенный HTTP-сервер, который дополняет стандартный HTTP-клиент и требует установки, как описано в *разделе по установке*.
- icu-date – библиотека форматирования даты и времени для Tarantool, которая основана на библиотеке для работы с Unicode (International Components for Unicode);

- kafka – это полноценная высокопроизводительная библиотека `kafka` для Tarantool на основе `librdkafka`;
- luacheck is a static analyzer and linter for Lua, preconfigured for Tarantool.
- luarapidxml – быстрый анализатор XML.
- luatest – среда тестирования Tarantool, которая написана на Lua.
- *membership* builds a mesh from multiple Tarantool instances based on gossip protocol. The mesh monitors itself, helps members discover everyone else in the group and get notified about their status changes with low latency. It is built upon the ideas from Consul or, more precisely, the SWIM algorithm.
- *metrics* is a collection of useful monitoring metrics.
- tracing is a module for debugging performance issues.
- *vshard* is an automatic sharding system that enables horizontal scaling for Tarantool DBMS instances.

### 8.12.2 Модули с закрытым исходным кодом

- ldap allows you to authenticate in a LDAP server and perform searches.
- odbc is an ODBC connector for Tarantool based on unixODBC.
- oracle is an Oracle connector for Lua applications through which they can send and receive data to and from Oracle databases. The advantage of the Tarantool-Oracle integration is that anyone can handle all the tasks with Oracle DBMSs (control, manipulation, storage, access) with the same high-level language (Lua) and with minimal delay.
- task is a module for managing background tasks in a Tarantool cluster.

### 8.12.3 Установка и использование модулей

Чтобы использовать модуль, установите следующие элементы:

1. Все необходимые сторонние пакеты программного обеспечения (при необходимости). Список программ см. в требованиях модуля.
2. Сам модуль на каждый экземпляр Tarantool:

```
$ tt rocks install MODULE_NAME [MODULE_VERSION]
```

See the *tt rocks reference* to learn more about managing Lua modules.

ГЛАВА 9

Справочники

## 9.1 Configuration reference

This topic describes all *configuration parameters* provided by Tarantool.

Most of the configuration options described in this reference can be applied to a specific instance, replica set, group, or to all instances globally. To do so, you need to define the required option at the *specified level*.

### 9.1.1 audit_log

**Enterprise Edition**

Configuring `audit_log` parameters is available in the Enterprise Edition only.

The `audit_log` section defines configuration parameters related to *audit logging*.

**Примечание:** `audit_log` can be defined in any *scope*.

- *audit_log.extract_key*
- *audit_log.file*
- *audit_log.filter*
- *audit_log.format*
- *audit_log.nonblock*
- *audit_log.pipe*
- *audit_log.spaces*
- *audit_log.to*

- *audit_log.syslog.**
  - *audit_log.syslog.facility*
  - *audit_log.syslog.identity*
  - *audit_log.syslog.server*

`audit_log.extract_key`
**Since:** *3.0.0*.

If set to `true`, the audit subsystem extracts and prints only the primary key instead of full tuples in DML events (`space_insert`, `space_replace`, `space_delete`). Otherwise, full tuples are logged. The option may be useful in case tuples are big.

Type: boolean
Default: false
Environment variable: TT_AUDIT_LOG_EXTRACT_KEY

`audit_log.file`
Specify a file for the audit log destination. You can set the `file` type using the *audit_log.to* option. If you write logs to a file, Tarantool reopens the audit log at SIGHUP.

Type: string
Default: „var/log/{{ instance_name }}/audit.log“
Environment variable: TT_AUDIT_LOG_FILE

`audit_log.filter`
Enable logging for a specified subset of audit events. This option accepts the following values:

- Event names (for example, `password_change`). For details, see *Audit log events*.
- Event groups (for example, `audit`). For details, see *Event groups*.

The option contains either one value from `Possible values` section (see below) or a combination of them.

To enable *custom audit log events*, specify the `custom` value in this option.

**Example**

```
filter: [ user_create,data_operations,ddl,custom ]
```

Type: array
Possible values: „all“, „audit“, „auth“, „priv“, „ddl“, „dml“, „data_operations“, „compatibility“, „audit_enable“, „auth_ok“, „auth_fail“, „disconnect“, „user_create“, „user_drop“, „role_create“, „role_drop“, „user_disable“, „user_enable“, „user_grant_rights“, „role_grant_rights“, „role_revoke_rights“, „password_change“, „access_denied“, „eval“, „call“, „space_select“,

„space_create“, „space_alter“, „space_drop“, „space_insert“, „space_replace“, „space_delete“, „custom“

Default: „nil“

Environment variable: TT_AUDIT_LOG_FILTER

`audit_log.format`

Specify a format that is used for the audit log.

**Example**

If you set the option to `plain`,

```
audit_log:
  to: file
  format: plain
```

the output in the file might look as follows:

```
2024-01-17T00:12:27.155+0300
4b5a2624-28e5-4b08-83c7-035a0c5a1db9
INFO remote:unix/:(socket)
session_type:console
module:tarantool
user:admin
type:space_create
tag:
description:Create space Bands
```

Type: string

Possible values: „json“, „csv“, „plain“

Default: „json“

Environment variable: TT_AUDIT_LOG_FORMAT

`audit_log.nonblock`

Specify the logging behavior if the system is not ready to write. If set to `true`, Tarantool does not block during logging if the system is non-writable and writes a message instead. Using this value may improve logging performance at the cost of losing some log messages.

---

**Примечание:** The option only has an effect if the *audit_log.to* is set to `syslog` or `pipe`.

---

Type: boolean

Default: false

Environment variable: TT_AUDIT_LOG_NONBLOCK

`audit_log.pipe`

Specify a pipe for the audit log destination. You can set the `pipe` type using the *audit_log.to* option. If log is a program, its pid is stored in the `audit.pid` field. You need to send it a signal to rotate logs.

**Example**

This starts the cronolog program when the server starts and sends all `audit_log` messages to cronolog standard input (`stdin`).

```
audit_log:
  to: pipe
  pipe: 'cronolog audit_tarantool.log'
```

Type: string

Default: box.NULL

Environment variable: TT_AUDIT_LOG_PIPE

`audit_log.spaces`

**Since:** *3.0.0*.

The array of space names for which data operation events (`space_select`, `space_insert`, `space_replace`, `space_delete`) should be logged. The array accepts string values. If set to *box.NULL*, the data operation events are logged for all spaces.

**Example**

In the example, only the events of `bands` and `singers` spaces are logged:

```
audit_log:
  spaces: [bands, singers]
```

Type: array

Default: box.NULL

Environment variable: TT_AUDIT_LOG_SPACES

`audit_log.to`

Enable audit logging and define the log location. This option accepts the following values:

- `devnull`: disable audit logging.
- `file`: write audit logs to a file (see *audit_log.file*).
- `pipe`: start a program and write audit logs to it (see *audit_log.pipe*).
- `syslog`: write audit logs to a system logger (see *audit_log.syslog.**).

By default, audit logging is disabled.

**Example**

The basic audit log configuration might look as follows:

```
audit_log:
  to: file
  file: 'audit_tarantool.log'
  filter: [ user_create,data_operations,ddl,custom ]
  format: json
  spaces: [ bands ]
  extract_key: true
```

Type: string

Possible values: „devnull“, „file“, „pipe“, „syslog“

Default: „devnull“

Environment variable: TT_AUDIT_LOG_TO

### audit_log.syslog.*

`audit_log.syslog.facility`

Specify a system logger keyword that tells syslogd where to send the message. You can enable logging to a system logger using the *audit_log.to* option.

See also: *syslog configuration example*.

Type: string

Possible values: „auth“, „authpriv“, „cron“, „daemon“, „ftp“, „kern“, „lpr“, „mail“, „news“, „security“, „syslog“, „user“, „uucp“, „local0“, „local1“, „local2“, „local3“, „local4“, „local5“, „local6“, „local7“

Default: „local7“

Environment variable: TT_AUDIT_LOG_SYSLOG_FACILITY

`audit_log.syslog.identity`

Specify an application name to show in logs. You can enable logging to a system logger using the *audit_log.to* option.

See also: *syslog configuration example*.

Type: string

Default: „tarantool“

Environment variable: TT_AUDIT_LOG_SYSLOG_IDENTITY

`audit_log.syslog.server`

Set a location for the syslog server. It can be a Unix socket path starting with „unix:“ or an ipv4 port number. You can enable logging to a system logger using the *audit_log.to* option.

**Example**

```
audit_log:
  to: syslog
  syslog:
    server: 'unix:/dev/log'
    facility: 'user'
    identity: 'tarantool_audit'
```

- *audit_log.syslog.server* – a syslog server location.
- *audit_log.syslog.facility* – a system logger keyword that tells syslogd where to send the message. The default value is `local7`.
- *audit_log.syslog.identity* – an application name to show in logs. The default value is `tarantool`.

These options are interpreted as a message for the syslogd program, which runs in the background of any Unix-like platform.

An example of a Tarantool audit log entry in the syslog:

```
09:32:52 tarantool_audit: {"time": "2024-02-08T09:32:52.190+0300", "uuid":
↪"94454e46-9a0e-493a-bb9f-d59e44a43581", "severity": "INFO", "remote": "unix/
↪:(socket)", "session_type": "console", "module": "tarantool", "user": "admin
↪", "type": "space_create", "tag": "", "description": "Create space bands"}
```

**Предупреждение:** Above is an example of writing audit logs to a directory shared with the system logs. Tarantool allows this option, but it is not recommended to do this to avoid difficulties when working with audit logs. System and audit logs should be written separately. To do this, create separate paths and specify them.

Type: string
Default: box.NULL
Environment variable: TT_AUDIT_LOG_SYSLOG_SERVER

### 9.1.2 config

The `config` section defines various parameters related to centralized configuration.

**Примечание:** `config` can be defined in the global *scope* only.

- *config.reload*
- *config.context.**
- *config.etcd.**
- *config.storage.**

`config.reload`
**Since:** *3.0.0*.

Specify how the configuration is reloaded. This option accepts the following values:

- `auto`: configuration is reloaded automatically when it is changed.
- `manual`: configuration should be reloaded manually. In this case, you can reload the configuration in the application code using *config:reload()*.

See also: *Reloading configuration*.

Type: string
Possible values: „auto“, „manual“
Default: „auto“
Environment variable: TT_CONFIG_RELOAD

### config.context.*

This section describes options related to loading configuration settings from external storage such as external files or environment variables.


- *config.context*
  - *config.context.<name>*
    - *config.context.<name>.env*
    - *config.context.<name>.from*
    - *config.context.<name>.file*
    - *config.context.<name>.env*


`config.context`
**Since:** *3.0.0*.

Specify how to load settings from external storage. For example, this option can be used to load passwords from safe storage. You can find examples in the *Loading secrets from safe storage* section.

Type: map
Default: nil
Environment variable: TT_CONFIG_CONTEXT

`config.context.<name>`
The name of an entity that identifies a configuration value to load.

`config.context.<name>.env`
The name of an environment variable to load a configuration value from. To load a configuration value from an environment variable, set *config.context.<name>.from* to `env`.

**Example:**

In this example, passwords are loaded from the `DBADMIN_PASSWORD` and `SAMPLEUSER_PASSWORD` environment variables:

```
config:
  context:
    dbadmin_password:
      from: env
      env: DBADMIN_PASSWORD
    sampleuser_password:
      from: env
      env: SAMPLEUSER_PASSWORD
```

See also: *Loading secrets from safe storage*.

`config.context.<name>.from`
: The type of storage to load a configuration value from. There are the following storage types:

  - `file`: load a configuration value from a file. In this case, you need to specify the path to the file using *config.context.<name>.file*.
  - `env`: load a configuration value from an environment variable. In this case, specify the environment variable name using *config.context.<name>.env*.

`config.context.<name>.file`
: The path to a file to load a configuration value from. To load a configuration value from a file, set *config.context.<name>.from* to `file`.

  **Example:**

  In this example, passwords are loaded from the `dbadmin_password.txt` and `sampleuser_password.txt` files:

```
config:
  context:
    dbadmin_password:
      from: file
      file: secrets/dbadmin_password.txt
      rstrip: true
    sampleuser_password:
      from: file
      file: secrets/sampleuser_password.txt
      rstrip: true
```

  See also: *Loading secrets from safe storage*.

`config.context.<name>.rstrip`
: (Optional) Whether to strip whitespace characters and newlines from the end of data.

### config.etcd.*

**Enterprise Edition**

Centralized configuration storages are supported by the Enterprise Edition only.

This section describes options related to providing connection settings to a *centralized etcd-based storage*.

- *config.etcd.endpoints*
- *config.etcd.prefix*

- *config.etcd.username*
- *config.etcd.password*
- *config.etcd.ssl.ca_file*
- *config.etcd.ssl.ca_path*
- *config.etcd.ssl.ssl_key*
- *config.etcd.ssl.verify_host*
- *config.etcd.ssl.verify_peer*
- *config.etcd.http.request.timeout*
- *config.etcd.http.request.unix_socket*

`config.etcd.endpoints`

**Since:** *3.0.0*.

The list of endpoints used to access an etcd server.

See also: *Configuring connection to an etcd storage*.

Type: array
Default: nil
Environment variable: TT_CONFIG_ETCD_ENDPOINTS

`config.etcd.prefix`

**Since:** *3.0.0*.

A key prefix used to search a configuration on an etcd server. Tarantool searches keys by the following path: `<prefix>/config/*`. Note that `<prefix>` should start with a slash (/).

See also: *Configuring connection to an etcd storage*.

Type: string
Default: nil
Environment variable: TT_CONFIG_ETCD_PREFIX

`config.etcd.username`

**Since:** *3.0.0*.

A username used for authentication.

See also: *Configuring connection to an etcd storage*.

Type: string
Default: nil
Environment variable: TT_CONFIG_ETCD_USERNAME

`config.etcd.password`
**Since:** *3.0.0*.

A password used for authentication.

See also: *Configuring connection to an etcd storage*.

Type: string
Default: nil
Environment variable: TT_CONFIG_ETCD_PASSWORD

`config.etcd.ssl.ca_file`
**Since:** *3.0.0*.

A path to a trusted certificate authorities (CA) file.

Type: string
Default: nil
Environment variable: TT_CONFIG_ETCD_SSL_CA_FILE

`config.etcd.ssl.ca_path`
**Since:** *3.0.0*.

A path to a directory holding certificates to verify the peer with.

Type: string
Default: nil
Environment variable: TT_CONFIG_ETCD_SSL_CA_PATH

`config.etcd.ssl.ssl_key`
**Since:** *3.0.0*.

A path to a private SSL key file.

Type: string
Default: nil
Environment variable: TT_CONFIG_ETCD_SSL_SSL_KEY

`config.etcd.ssl.verify_host`
**Since:** *3.0.0*.

Enable verification of the certificate's name (CN) against the specified host.

Type: boolean
Default: nil
Environment variable: TT_CONFIG_ETCD_SSL_VERIFY_HOST

`config.etcd.ssl.verify_peer`
**Since:** *3.0.0*.

Enable verification of the peer's SSL certificate.

Type: boolean
Default: nil
Environment variable: TT_CONFIG_ETCD_SSL_VERIFY_PEER

`config.etcd.http.request.timeout`
**Since:** *3.0.0*.

A time period required to process an HTTP request to an etcd server: from sending a request to receiving a response.

See also: *Configuring connection to an etcd storage*.

Type: number
Default: nil
Environment variable: TT_CONFIG_ETCD_HTTP_REQUEST_TIMEOUT

`config.etcd.http.request.unix_socket`
**Since:** *3.0.0*.

A Unix domain socket used to connect to an etcd server.

Type: string
Default: nil
Environment variable: TT_CONFIG_ETCD_HTTP_REQUEST_UNIX_SOCKET

### config.storage.*

**Enterprise Edition**

Centralized configuration storages are supported by the Enterprise Edition only.

This section describes options related to providing connection settings to a *centralized Tarantool-based storage*.

- *config.storage.endpoints*
- *config.storage.prefix*
- *config.storage.reconnect_after*
- *config.storage.timeout*

`config.storage.endpoints`

**Since:** *3.0.0*.

An array of endpoints used to access a configuration storage. Each endpoint can include the following fields:

- `uri`: a URI of the configuration storage's instance.
- `login`: a username used to connect to the instance.
- `password`: a password used for authentication.
- `params`: SSL parameters required for *encrypted connections* (*<uri>.params.**).

See also: *Configuring connection to a Tarantool storage*.

Type: array
Default: nil
Environment variable: TT_CONFIG_STORAGE_ENDPOINTS

`config.storage.prefix`

**Since:** *3.0.0*.

A key prefix used to search a configuration in a centralized configuration storage. Tarantool searches keys by the following path: `<prefix>/config/*`. Note that `<prefix>` should start with a slash (/).

See also: *Configuring connection to a Tarantool storage*.

Type: string
Default: nil
Environment variable: TT_CONFIG_STORAGE_PREFIX

`config.storage.reconnect_after`

**Since:** *3.0.0*.

A number of seconds to wait before reconnecting to a configuration storage.

Type: number
Default: 3
Environment variable: TT_CONFIG_STORAGE_RECONNECT_AFTER

`config.storage.timeout`
**Since:** *3.0.0*.

The interval (in seconds) to perform the status check of a configuration storage.

See also: *Configuring connection to a Tarantool storage*.

Type: number
Default: 3
Environment variable: TT_CONFIG_STORAGE_TIMEOUT

### 9.1.3 credentials

The `credentials` section allows you to create users and grant them the specified privileges. Learn more in *Credentials*.

**Примечание:** `credentials` can be defined in any *scope*.

- *credentials.roles.**
- *credentials.users.**
- *<user_or_role_name>.privileges.**

#### credentials.roles.*

- *credentials.roles*
  - *credentials.roles.<role_name>.roles*
  - *credentials.roles.<role_name>.privileges*

`credentials.roles`
An array of *roles* that can be granted to users or other roles.

**Example:**

In the example below, the `writers_space_reader` role gets privileges to select data in the `writers` space:

```
roles:
  writers_space_reader:
    privileges:
    - permissions: [ read ]
      spaces: [ writers ]
```

See also: *Managing users and roles*.

Type: map
Default: nil
Environment variable: TT_CREDENTIALS_ROLES

`credentials.roles.<role_name>.roles`
An array of *roles* granted to this role.

`credentials.roles.<role_name>.privileges`
An array of *privileges* granted to this role.

See *<user_or_role_name>.privileges.**.

### credentials.users.*

- *credentials.users*
  - *credentials.users.<username>.password*
  - *credentials.users.<username>.roles*
  - *credentials.users.<username>.privileges*

`credentials.users`
An array of *users*.

**Example:**

In this example, `sampleuser` gets the following privileges:

- Privileges granted to the `writers_space_reader` role.
- Privileges to select and modify data in the `books` space.

```
sampleuser:
  password: '123456'
  roles: [ writers_space_reader ]
  privileges:
  - permissions: [ read, write ]
    spaces: [ books ]
```

See also: *Managing users and roles*.

Type: map
Default: nil
Environment variable: TT_CREDENTIALS_USERS

`credentials.users.<username>.password`
A user's password.

**Example:**

In the example below, a password for the `dbadmin` user is set:

```
credentials:
  users:
    dbadmin:
      password: 'TOp_Secret_P@$$wOrd'
```

See also: *Loading secrets from safe storage*.

`credentials.users.<username>.roles`
An array of *roles* granted to this user.

`credentials.users.<username>.privileges`
An array of *privileges* granted to this user.

See *<user_or_role_name>.privileges.**.

#### <user_or_role_name>.privileges.*

- *<user_or_role_name>.privileges*
  - *<user_or_role_name>.privileges.permissions*
  - *<user_or_role_name>.privileges.spaces*
  - *<user_or_role_name>.privileges.functions*
  - *<user_or_role_name>.privileges.sequences*
  - *<user_or_role_name>.privileges.lua_eval*
  - *<user_or_role_name>.privileges.lua_call*
  - *<user_or_role_name>.privileges.sql*

`<user_or_role_name>.privileges`
Privileges that can be granted to a user or role using the following options:

- *credentials.users.<username>.privileges*
- *credentials.roles.<role_name>.privileges*

`<user_or_role_name>.privileges.permissions`
*Permissions* assigned to this user or a user with this role.

**Example:**

In this example, `sampleuser` gets privileges to select and modify data in the `books` space:

```
sampleuser:
  password: '123456'
  roles: [ writers_space_reader ]
  privileges:
  - permissions: [ read, write ]
    spaces: [ books ]
```

See also: *Managing users and roles*.

`<user_or_role_name>.privileges.spaces`
Spaces to which this user or a user with this role gets the specified permissions.

**Example:**

In this example, `sampleuser` gets privileges to select and modify data in the `books` space:

```
sampleuser:
  password: '123456'
  roles: [ writers_space_reader ]
  privileges:
  - permissions: [ read, write ]
    spaces: [ books ]
```

See also: *Managing users and roles*.

`<user_or_role_name>.privileges.functions`
: Functions to which this user or a user with this role gets the specified permissions.

`<user_or_role_name>.privileges.sequences`
: Sequences to which this user or a user with this role gets the specified permissions.

`<user_or_role_name>.privileges.lua_eval`
: Whether this user or a user with this role can execute arbitrary Lua code.

`<user_or_role_name>.privileges.lua_call`
: Whether this user or a user with this role can call any global user-defined Lua function.

`<user_or_role_name>.privileges.sql`
: Whether this user or a user with this role can execute an arbitrary SQL expression.

### 9.1.4 database

The `database` section defines database-specific configuration parameters, such as an instance's read-write mode or transaction isolation level.

---

**Примечание:** `database` can be defined in any *scope*.

---

- *database.hot_standby*
- *database.instance_uuid*
- *database.mode*
- *database.replicaset_uuid*
- *database.txn_isolation*
- *database.txn_timeout*
- *database.use_mvcc_engine*

`database.hot_standby`

Type: boolean

Default: false

Environment variable: TT_DATABASE_HOT_STANDBY

`database.instance_uuid`
: An *instance UUID*.

By default, instance UUIDs are generated automatically. `database.instance_uuid` can be used to specify an instance identifier manually.

UUIDs should follow these rules:

- The values must be true unique identifiers, not shared by other instances or replica sets within the common infrastructure.
- The values must be used consistently, not changed after the initial setup. The initial values are stored in *snapshot files* and are checked whenever the system is restarted.
- The values must comply with RFC 4122. The nil UUID is not allowed.

See also: *database.replicaset_uuid*

Type: string
Default: *box.NULL*
Environment variable: TT_DATABASE_INSTANCE_UUID

`database.mode`

An instance's operating mode. This option is in effect if *replication.failover* is set to `off`.

The following modes are available:

- `rw`: an instance is in read-write mode.
- `ro`: an instance is in read-only mode.

If not specified explicitly, the default value depends on the number of instances in a replica set. For a single instance, the `rw` mode is used, while for multiple instances, the `ro` mode is used.

**Example**

You can set the `database.mode` option to `rw` on all instances in a replica set to make a *master-master* configuration. In this case, `replication.failover` should be set to `off`.

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]

iproto:
  advertise:
    peer:
      login: replicator

replication:
  failover: off

groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            database:
              mode: rw
            iproto:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            database:
              mode: rw
            iproto:
              listen:
              - uri: '127.0.0.1:3302'

# Load sample data
app:
  file: 'myapp.lua'
```

Type: string
Default: *box.NULL* (the actual default value depends on the number of instances in a replica set)
Environment variable: TT_DATABASE_MODE

`database.replicaset_uuid`
A *replica set UUID*.

By default, replica set UUIDs are generated automatically. `database.replicaset_uuid` can be used to specify a replica set identifier manually.

See also: *database.instance_uuid*

Type: string
Default: *box.NULL*
Environment variable: TT_DATABASE_REPLICASET_UUID

`database.txn_isolation`
A transaction *isolation level*.

Type: string
Default: `best-effort`
Possible values: `best-effort`, `read-committed`, `read-confirmed`
Environment variable: TT_DATABASE_TXN_ISOLATION

`database.txn_timeout`
A timeout (in seconds) after which the transaction is rolled back.

See also: *box.begin()*

Type: number
Default: 3153600000 (~100 years)
Environment variable: TT_DATABASE_TXN_TIMEOUT

`database.use_mvcc_engine`
Whether the *transactional manager* is enabled.

Type: boolean
Default: false
Environment variable: TT_DATABASE_USE_MVCC_ENGINE

### 9.1.5 flightrec

---

**Enterprise Edition**

Configuring `flightrec` parameters is available in the Enterprise Edition only.

---

The `flightrec` section describes options related to the *flight recorder* configuration.

---

**Примечание:** `flightrec` can be defined in any *scope*.

---

- *flightrec.enabled*
- *flightrec.logs_size*
- *flightrec.logs_max_msg_size*
- *flightrec.logs_log_level*
- *flightrec.metrics_period*
- *flightrec.metrics_interval*
- *flightrec.requests_size*
- *flightrec.requests_max_req_size*
- *flightrec.requests_max_res_size*

`flightrec.enabled`
Enable the *flight recorder*.

Type: boolean
Default: false
Environment variable: TT_FLIGHTREC_ENABLED

`flightrec.logs_size`
Specify the size (in bytes) of the log storage. You can set this option to `0` to disable the log storage.

Type: integer
Default: 10485760
Environment variable: TT_FLIGHTREC_LOGS_SIZE

`flightrec.logs_max_msg_size`
Specify the maximum size (in bytes) of the log message. The log message is truncated if its size exceeds this limit.

Type: integer
Default: 4096
Maximum: 16384
Environment variable: TT_FLIGHTREC_LOGS_MAX_MSG_SIZE

`flightrec.logs_log_level`
Specify the level of detail the log has. The default value is 6 (`VERBOSE`). You can learn more about log levels from the *log_level* option description. Note that the `flightrec.logs_log_level` value might differ from `log_level`.

Type: integer
Default: 6
Environment variable: TT_FLIGHTREC_LOGS_LOG_LEVEL

`flightrec.metrics_period`
Specify the time period (in seconds) that defines how long metrics are stored from the moment of dump. So, this value defines how much historical metrics data is collected up to the moment of crash. The frequency of metric dumps is defined by *flightrec.metrics_interval*.

Type: integer
Default: 180
Environment variable: TT_FLIGHTREC_METRICS_PERIOD

`flightrec.metrics_interval`
Specify the time interval (in seconds) that defines the frequency of dumping metrics. This value shouldn't exceed *flightrec.metrics_period*.

Type: number
Default: 1.0
Minimum: 0.001
Environment variable: TT_FLIGHTREC_METRICS_INTERVAL

---

**Примечание:** Given that the average size of a metrics entry is 2 kB, you can estimate the size of the metrics storage as follows:

```
(flightrec_metrics_period / flightrec_metrics_interval) * 2 kB
```

`flightrec.requests_size`
: Specify the size (in bytes) of storage for the request and response data. You can set this parameter to `0` to disable a storage of requests and responses.

  Type: integer
  Default: 10485760
  Environment variable: TT_FLIGHTREC_REQUESTS_SIZE

`flightrec.requests_max_req_size`
: Specify the maximum size (in bytes) of a request entry. A request entry is truncated if this size is exceeded.

  Type: integer
  Default: 16384
  Environment variable: TT_FLIGHTREC_REQUESTS_MAX_REQ_SIZE

`flightrec.requests_max_res_size`
: Specify the maximum size (in bytes) of a response entry. A response entry is truncated if this size is exceeded.

  Type: integer
  Default: 16384
  Environment variable: TT_FLIGHTREC_REQUESTS_MAX_RES_SIZE

### 9.1.6 iproto

The `iproto` section is used to configure parameters related to *communicating to and between cluster instances*.

**Примечание:** `iproto` can be defined in any *scope*.

- *iproto.advertise.**
  - *iproto.advertise.client*
  - *iproto.advertise.peer*
  - *iproto.advertise.sharding*
  - *iproto.advertise.<peer_or_sharding>.**
- *iproto.**
  - *iproto.listen*
  - *iproto.net_msg_max*

  - *iproto.readahead*
  - *iproto.threads*
- *<uri>.params.**

### iproto.advertise.*

`iproto.advertise.client`
A URI used to advertise the current instance to clients.

The `iproto.advertise.client` option accepts a URI in the following formats:

- An address: `host:port`.
- A Unix domain socket: `unix/:`.

Note that this option doesn't allow to set a username and password. If a remote client needs this information, it should be delivered outside of the cluster configuration.

---

**Примечание:** The host value cannot be `0.0.0.0`/`[::]` and the port value cannot be `0`.

---

Type: string
Default: *box.NULL*
Environment variable: TT_IPROTO_ADVERTISE_CLIENT

`iproto.advertise.peer`
Settings used to advertise the current instance to other cluster members. The format of these settings is described in *iproto.advertise.<peer_or_sharding>.**.

**Example**

In the example below, the following configuration options are specified:

- In the *credentials* section, the `replicator` user with the `replication` role is created.
- `iproto.advertise.peer` specifies that other instances should connect to an address defined in *iproto.listen* using the `replicator` user.

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]

iproto:
  advertise:
    peer:
      login: replicator

replication:
  failover: election
```

(continues on next page)

(продолжение с предыдущей страницы)

```
groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
```

Type: map

Environment variable: see *iproto.advertise.<peer_or_sharding>.**

`iproto.advertise.sharding`

Settings used to advertise the current instance to a router and rebalancer. The format of these settings is described in *iproto.advertise.<peer_or_sharding>.**.

---

**Примечание:** If `iproto.advertise.sharding` is not specified, advertise settings from *iproto.advertise.peer* are used.

---

**Example**

In the example below, the following configuration options are specified:

- In the *credentials* section, the `replicator` and `storage` users are created.
- `iproto.advertise.peer` specifies that other instances should connect to an address defined in *iproto.listen* with the `replicator` user.
- `iproto.advertise.sharding` specifies that a router should connect to storages using an address defined in *iproto.listen* with the `storage` user.

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]
    storage:
      password: 'secret'
      roles: [sharding]

iproto:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
advertise:
  peer:
    login: replicator
  sharding:
    login: storage
```

Type: map
Environment variable: see *iproto.advertise.<peer_or_sharding>.**

### iproto.advertise.<peer_or_sharding>.*

`iproto.advertise.<peer_or_sharding>.uri`
(Optional) A URI used to advertise the current instance. By default, the URI defined in *iproto.listen* is used to advertise the current instance.

---

**Примечание:** The host value cannot be `0.0.0.0`/`[::]` and the port value cannot be `0`.

---

Type: string
Default: nil
Environment variable: TT_IPROTO_ADVERTISE_PEER_URI, TT_IPROTO_ADVERTISE_SHARDING_URI

`iproto.advertise.<peer_or_sharding>.login`
(Optional) A username used to connect to the current instance. If a username is not set, the `guest` user is used.

Type: string
Default: nil
Environment variable: TT_IPROTO_ADVERTISE_PEER_LOGIN, TT_IPROTO_ADVERTISE_SHARDING_LOGIN

`iproto.advertise.<peer_or_sharding>.password`
(Optional) A password for the specified user. If a `login` is specified but a password is missing, it is taken from the user's *credentials*.

Type: string
Default: nil
Environment variable: TT_IPROTO_ADVERTISE_PEER_PASSWORD, TT_IPROTO_ADVERTISE_SHARDING_PASSWORD

`iproto.advertise.<peer_or_sharding>.params`
: (Optional) URI parameters (*<uri>.params.**) required for connecting to the current instance.

### iproto.*

`iproto.listen`
: An array of URIs used to listen for incoming requests. If required, you can enable SSL for specific URIs by providing additional parameters (*<uri>.params.**).

  Note that a URI value can't contain parameters, a login, or a password.

  **Example**

  In the example below, `iproto.listen` is set explicitly for each instance in a cluster:

```
groups:
  group001:
    replicasets:
      replicaset001:
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
```

  See also: *Connections*.

  Type: array
  Default: nil
  Environment variable: TT_IPROTO_LISTEN

`iproto.net_msg_max`
: To handle messages, Tarantool allocates *fibers*. To prevent fiber overhead from affecting the whole system, Tarantool restricts how many messages the fibers handle, so that some pending requests are blocked.

  - On powerful systems, increase `net_msg_max`, and the scheduler starts processing pending requests immediately.
  - On weaker systems, decrease `net_msg_max`, and the overhead may decrease. However, this may take some time because the scheduler must wait until already-running requests finish.

  When `net_msg_max` is reached, Tarantool suspends processing of incoming packages until it has processed earlier messages. This is not a direct restriction of the number of fibers that handle network messages, rather it is a system-wide restriction of channel bandwidth. This in turn restricts the number

of incoming network messages that the *transaction processor thread* handles, and therefore indirectly affects the fibers that handle network messages.

---

**Примечание:** The number of fibers is smaller than the number of messages because messages can be released as soon as they are delivered, while incoming requests might not be processed until some time after delivery.

---

Type: integer

Default: 768

Environment variable: TT_IPROTO_NET_MSG_MAX

`iproto.readahead`

The size of the read-ahead buffer associated with a client connection. The larger the buffer, the more memory an active connection consumes, and the more requests can be read from the operating system buffer in a single system call.

The recommendation is to make sure that the buffer can contain at least a few dozen requests. Therefore, if a typical tuple in a request is large, e.g. a few kilobytes or even megabytes, the read-ahead buffer size should be increased. If batched request processing is not used, it's prudent to leave this setting at its default.

Type: integer

Default: 16320

Environment variable: TT_IPROTO_READAHEAD

`iproto.threads`

The number of *network threads*. There can be unusual workloads where the network thread is 100% loaded and the transaction processor thread is not, so the network thread is a bottleneck. In that case, set `iproto_threads` to 2 or more. The operating system kernel determines which connection goes to which thread.

Type: integer

Default: 1

Environment variable: TT_IPROTO_THREADS

### <uri>.params.*

**Enterprise Edition**

TLS traffic encryption is supported by the Enterprise Edition only.

URI parameters that can be used in the following options:

- *iproto.advertise.<peer_or_sharding>.params*
- *iproto.listen.<uri>.params*

**Примечание:** Note that `<uri>.params.*` options don't have corresponding *environment variables* for URIs specified in `iproto.listen`.

`<uri>.params.transport`
: Allows you to enable *traffic encryption* for client-server communications over binary connections. In a Tarantool cluster, one instance might act as the server that accepts connections from other instances and the client that connects to other instances.

  `<uri>.params.transport` accepts one of the following values:

  - `plain` (default): turn off traffic encryption.
  - `ssl`: encrypt traffic by using the TLS 1.2 protocol (Enterprise Edition only).

  **Example**

  The example below demonstrates how to enable traffic encryption by using a self-signed server certificate. The following parameters are specified for each instance:

  - `ssl_cert_file`: a path to an SSL certificate file.
  - `ssl_key_file`: a path to a private SSL key file.

```
replicaset001:
  replication:
    failover: manual
  leader: instance001
  iproto:
    advertise:
      peer:
        login: replicator
  instances:
    instance001:
      iproto:
        listen:
        - uri: '127.0.0.1:3301'
          params:
            transport: 'ssl'
            ssl_cert_file: 'certs/server.crt'
            ssl_key_file: 'certs/server.key'
    instance002:
      iproto:
        listen:
        - uri: '127.0.0.1:3302'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        params:
          transport: 'ssl'
          ssl_cert_file: 'certs/server.crt'
          ssl_key_file: 'certs/server.key'
    instance003:
      iproto:
        listen:
        - uri: '127.0.0.1:3303'
          params:
            transport: 'ssl'
            ssl_cert_file: 'certs/server.crt'
            ssl_key_file: 'certs/server.key'
```

You can find the full example here: ssl_without_ca.

Type: string

Default: „plain“

Environment variable: TT_IPROTO_ADVERTISE_PEER_PARAMS_TRANSPORT, TT_IPROTO_ADVERTISE_SHARDING_PARAMS_TRANSPORT

`<uri>.params.ssl_ca_file`

(Optional) A path to a trusted certificate authorities (CA) file. If not set, the peer won't be checked for authenticity.

Both a server and a client can use the `ssl_ca_file` parameter:

- If it's on the server side, the server verifies the client.
- If it's on the client side, the client verifies the server.
- If both sides have the CA files, the server and the client verify each other.

**See also:** *<uri>.params.transport*.

Type: string

Default: nil

Environment variable: TT_IPROTO_ADVERTISE_PEER_PARAMS_SSL_CA_FILE, TT_IPROTO_ADVERTISE_SHARDING_PARAMS_SSL_CA_FILE

`<uri>.params.ssl_cert_file`

A path to an SSL certificate file:

- For a server, it's mandatory.
- For a client, it's mandatory if the *ssl_ca_file* parameter is set for a server; otherwise, optional.

**See also:** *<uri>.params.transport*.

Type: string
Default: nil
Environment variable: TT_IPROTO_ADVERTISE_PEER_PARAMS_SSL_CERT_FILE, TT_IPROTO_ADVERTISE_SHARDING_PARAMS_SSL_CERT_FILE

`<uri>.params.ssl_ciphers`
(Optional) A colon-separated (:) list of SSL cipher suites the connection can use. Note that the list is not validated: if a cipher suite is unknown, Tarantool ignores it, doesn't establish the connection, and writes to the log that no shared cipher was found.

The supported cipher suites are:

- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-RSA-AES256-GCM-SHA384
- DHE-RSA-AES256-GCM-SHA384
- ECDHE-ECDSA-CHACHA20-POLY1305
- ECDHE-RSA-CHACHA20-POLY1305
- DHE-RSA-CHACHA20-POLY1305
- ECDHE-ECDSA-AES128-GCM-SHA256
- ECDHE-RSA-AES128-GCM-SHA256
- DHE-RSA-AES128-GCM-SHA256
- ECDHE-ECDSA-AES256-SHA384
- ECDHE-RSA-AES256-SHA384
- DHE-RSA-AES256-SHA256
- ECDHE-ECDSA-AES128-SHA256
- ECDHE-RSA-AES128-SHA256
- DHE-RSA-AES128-SHA256
- ECDHE-ECDSA-AES256-SHA
- ECDHE-RSA-AES256-SHA
- DHE-RSA-AES256-SHA
- ECDHE-ECDSA-AES128-SHA
- ECDHE-RSA-AES128-SHA
- DHE-RSA-AES128-SHA
- AES256-GCM-SHA384
- AES128-GCM-SHA256
- AES256-SHA256
- AES128-SHA256
- AES256-SHA
- AES128-SHA
- GOST2012-GOST8912-GOST8912

- GOST2001-GOST89-GOST89

For detailed information on SSL ciphers and their syntax, refer to OpenSSL documentation.

**See also:** *<uri>.params.transport*.

Type: string
Default: nil
Environment variable: TT_IPROTO_ADVERTISE_PEER_PARAMS_SSL_CIPHERS, TT_IPROTO_ADVERTISE_SHARDING_PARAMS_SSL_CIPHERS

`<uri>.params.ssl_key_file`

A path to a private SSL key file:

- For a server, it's mandatory.
- For a client, it's mandatory if the *ssl_ca_file* parameter is set for a server; otherwise, optional.

If the private key is encrypted, provide a password for it in the `ssl_password` or `ssl_password_file` parameter.

**See also:** *<uri>.params.transport*.

Type: string
Default: nil
Environment variable: TT_IPROTO_ADVERTISE_PEER_PARAMS_SSL_KEY_FILE, TT_IPROTO_ADVERTISE_SHARDING_PARAMS_SSL_KEY_FILE

`<uri>.params.ssl_password`

(Optional) A password for an encrypted private SSL key provided using `ssl_key_file`. Alternatively, the password can be provided in `ssl_password_file`.

Tarantool applies the `ssl_password` and `ssl_password_file` parameters in the following order:

1. If `ssl_password` is provided, Tarantool tries to decrypt the private key with it.
2. If `ssl_password` is incorrect or isn't provided, Tarantool tries all passwords from `ssl_password_file` one by one in the order they are written.
3. If `ssl_password` and all passwords from `ssl_password_file` are incorrect, or none of them is provided, Tarantool treats the private key as unencrypted.

**See also:** *<uri>.params.transport*.

Type: string
Default: nil
Environment variable: TT_IPROTO_ADVERTISE_PEER_PARAMS_SSL_PASSWORD, TT_IPROTO_ADVERTISE_SHARDING_PARAMS_SSL_PASSWORD

`<uri>.params.ssl_password_file`
: (Optional) A text file with one or more passwords for encrypted private SSL keys provided using `ssl_key_file` (each on a separate line). Alternatively, the password can be provided in `ssl_password`.

  **See also:** *<uri>.params.transport*.

  Type: string

  Default: nil

  Environment variable: TT_IPROTO_ADVERTISE_PEER_PARAMS_SSL_PASSWORD_FILE, TT_IPROTO_ADVERTISE_SHARDING_PARAMS_SSL_PASSWORD_FILE

### 9.1.7 groups

The `groups` section provides the ability to define the *full topology of a Tarantool cluster*.

---

**Примечание:** `groups` can be defined in the global *scope* only.

---

- *groups.<group_name>*
- *groups.<group_name>.replicasets*
- *groups.<group_name>.<config_parameter>*

`groups.<group_name>`
: A group name.

`groups.<group_name>.replicasets`
: Replica sets that belong to this group. See *replicasets*.

`groups.<group_name>.<config_parameter>`
: Any configuration parameter that can be defined in the group *scope*. For example, *iproto* and *database* configuration parameters defined at the group level are applied to all instances in this group.

#### replicasets

---

**Примечание:** `replicasets` can be defined in the group *scope* only.

---

- *replicasets.<replicaset_name>*
- *replicasets.<replicaset_name>.leader*
- *replicasets.<replicaset_name>.bootstrap_leader*
- *replicasets.<replicaset_name>.instances*
- *replicasets.<replicaset_name>.<config_parameter>*

`replicasets.<replicaset_name>`
: A replica set name.

`replicasets.<replicaset_name>.leader`
: A replica set leader. This option can be used to set a replica set leader when `manual` *replication.failover* is used.

To perform *controlled failover*, `<replicaset_name>.leader` can be temporarily removed or set to `null`.

**Example**

```
replication:
  failover: manual

groups:
  group001:
    replicasets:
      replicaset001:
        leader: instance001
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
```

`replicasets.<replicaset_name>.bootstrap_leader`

A bootstrap leader for a replica set. To specify a bootstrap leader manually, you need to set *replication.bootstrap_strategy* to `config`.

**Example**

```
groups:
  group001:
    replicasets:
      replicaset001:
        replication:
          bootstrap_strategy: config
        bootstrap_leader: instance001
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
```

`replicasets.<replicaset_name>.instances`

Instances that belong to this replica set. See *instances*.

`replicasets.<replicaset_name>.<config_parameter>`
Any configuration parameter that can be defined in the replica set *scope*. For example, *iproto* and *database* configuration parameters defined at the replica set level are applied to all instances in this replica set.

#### instances

---

**Примечание:** `instances` can be defined in the replica set *scope* only.

---

- *instances.<instance_name>*
- *instances.<instance_name>.<config_parameter>*

`instances.<instance_name>`
An instance name.

`instances.<instance_name>.<config_parameter>`
Any configuration parameter that can be defined in the instance *scope*. For example, *iproto* and *database* configuration parameters defined at the instance level are applied to this instance only.

### 9.1.8 log

The `log` section defines configuration parameters related to logging. To handle logging in your application, use the *log module*.

---

**Примечание:** `log` can be defined in any *scope*.

---

- *log.to*
- *log.file*
- *log.format*
- *log.level*
- *log.modules*
- *log.nonblock*
- *log.pipe*
- *log.syslog.**
  - *log.syslog.facility*
  - *log.syslog.identity*
  - *log.syslog.server*

`log.to`
Define a location Tarantool sends logs to. This option accepts the following values:

- `stderr`: write logs to the standard error stream.
- `file`: write logs to a file (see *log.file*).
- `pipe`: start a program and write logs to its standard input (see *log.pipe*).

- `syslog`: write logs to a system logger (see *log.syslog.**).

Type: string
Default: „stderr“
Environment variable: TT_LOG_TO

`log.file`
Specify a file for logs destination. To write logs to a file, you need to set *log.to* to `file`. Otherwise, `log.file` is ignored.

**Example**

The example below shows how to write logs to a file placed in the specified directory:

```
log:
  to: file
  file: var/log/{{ instance_name }}/instance.log
```

Example on GitHub: log_file.

Type: string
Default: „var/log/{{ instance_name }}/tarantool.log“
Environment variable: TT_LOG_FILE

`log.format`
Specify a format that is used for a log entry. The following formats are supported:

- `plain`: a log entry is formatted as plain text. Example:

```
2024-04-09 11:00:10.369 [12089] main/104/interactive I> log level 5 (INFO)
```

- `json`: a log entry is formatted as JSON and includes additional fields. Example:

```
{
  "time": "2024-04-09T11:00:57.174+0300",
  "level": "INFO",
  "message": "log level 5 (INFO)",
  "pid": 12160,
  "cord_name": "main",
  "fiber_id": 104,
  "fiber_name": "interactive",
  "file": "src/main.cc",
  "line": 498
}
```

Type: string

Default: „plain“
Environment variable: TT_LOG_FORMAT

`log.level`
Specify the level of detail logs have. There are the following levels:

- 0 – `fatal`
- 1 – `syserror`
- 2 – `error`
- 3 – `crit`
- 4 – `warn`
- 5 – `info`
- 6 – `verbose`
- 7 – `debug`

By setting `log.level`, you can enable logging of all events with severities above or equal to the given level.

**Example**

The example below shows how to log all events with severities above or equal to the `VERBOSE` level.

```
log:
  level: 'verbose'
```

Example on GitHub: log_level.

Type: number, string
Default: 5
Environment variable: TT_LOG_LEVEL

`log.modules`
Configure the specified log levels (*log.level*) for different modules.

You can specify a logging level for the following module types:

- Modules (files) that use the default logger. Example: *Set log levels for files that use the default logger*.
- Modules that use custom loggers created using the *log.new()* function. Example: *Set log levels for modules that use custom loggers*.
- The `tarantool` module that enables you to configure the logging level for Tarantool core messages. Specifically, it configures the logging level for messages logged from non-Lua code, including C modules. Example: *Set a log level for C modules*.

**Example 1: Set log levels for files that use the default logger**

Suppose you have two identical modules placed by the following paths: `test/module1.lua` and `test/module2.lua`. These modules use the default logger and look as follows:

```
return {
    say_hello = function()
        local log = require('log')
        log.info('Info message from module1')
    end
}
```

To configure logging levels, you need to provide module names corresponding to paths to these modules:

```
log:
  modules:
    test.module1: 'verbose'
    test.module2: 'error'
app:
  file: 'app.lua'
```

To load these modules in your application (`app.lua`), you need to add the corresponding `require` directives:

```
module1 = require('test.module1')
module2 = require('test.module2')
```

Given that `module1` has the `verbose` logging level and `module2` has the `error` level, calling `module1.say_hello()` shows a message but `module2.say_hello()` is swallowed:

```
-- Prints 'info' messages --
module1.say_hello()
--[[
[92617] main/103/interactive/test.logging.module1 I> Info message from module1
---
...
--]]

-- Swallows 'info' messages --
module2.say_hello()
--[[
---
...
--]]
```

Example on GitHub: log_existing_modules.

**Example 2: Set log levels for modules that use custom loggers**

This example shows how to set the `verbose` level for `module1` and the `error` level for `module2`:

```
log:
  modules:
    module1: 'verbose'
    module2: 'error'
app:
  file: 'app.lua'
```

To create custom loggers in your application (`app.lua`), call the *log.new()* function:

```
-- Creates new loggers --
module1_log = require('log').new('module1')
module2_log = require('log').new('module2')
```

Given that `module1` has the `verbose` logging level and `module2` has the `error` level, calling `module1_log.info()` shows a message but `module2_log.info()` is swallowed:

```
-- Prints 'info' messages --
module1_log.info('Info message from module1')
--[[
[16300] main/103/interactive/module1 I> Info message from module1
---
...
--]]

-- Swallows 'debug' messages --
module1_log.debug('Debug message from module1')
--[[
---
...
--]]

-- Swallows 'info' messages --
module2_log.info('Info message from module2')
--[[
---
...
--]]
```

Example on GitHub: log_new_modules.

**Example 3: Set a log level for C modules**

This example shows how to set the `info` level for the `tarantool` module:

```
log:
  modules:
    tarantool: 'info'
app:
  file: 'app.lua'
```

The specified level affects messages logged from C modules:

```
ffi = require('ffi')

-- Prints 'info' messages --
ffi.C._say(ffi.C.S_INFO, nil, 0, nil, 'Info message from C module')
--[[
[6024] main/103/interactive I> Info message from C module
---
...
--]]

-- Swallows 'debug' messages --
```

(continues on next page)

(продолжение с предыдущей страницы)

```
ffi.C._say(ffi.C.S_DEBUG, nil, 0, nil, 'Debug message from C module')
--[[
---
...
--]]
```

The example above uses the LuaJIT ffi library to call C functions provided by the `say` module.

Example on GitHub: log_existing_c_modules.

Type: map
Default: *box.NULL*
Environment variable: TT_LOG_MODULES

`log.nonblock`

Specify the logging behavior if the system is not ready to write. If set to `true`, Tarantool does not block during logging if the system is non-writable and writes a message instead. Using this value may improve logging performance at the cost of losing some log messages.

---

**Примечание:** The option only has an effect if the *log.to* is set to `syslog` or `pipe`.

---

Type: boolean
Default: false
Environment variable: TT_LOG_NONBLOCK

`log.pipe`

Start a program and write logs to its standard input (`stdin`). To send logs to a program's standard input, you need to set *log.to* to `pipe`.

**Example**

In the example below, Tarantool writes logs to the standard input of `cronolog`:

```
log:
  to: pipe
  pipe: 'cronolog tarantool.log'
```

Example on GitHub: log_pipe.

Type: string
Default: *box.NULL*
Environment variable: TT_LOG_PIPE

### log.syslog.*

`log.syslog.facility`
Specify the syslog facility to be used when syslog is enabled. To write logs to syslog, you need to set *log.to* to `syslog`.

Type: string
Possible values: „auth“, „authpriv“, „cron“, „daemon“, „ftp“, „kern“, „lpr“, „mail“, „news“, „security“, „syslog“, „user“, „uucp“, „local0“, „local1“, „local2“, „local3“, „local4“, „local5“, „local6“, „local7“
Default: „local7“
Environment variable: TT_LOG_SYSLOG_FACILITY

`log.syslog.identity`
Specify an application name used to identify Tarantool messages in syslog logs. To write logs to syslog, you need to set *log.to* to `syslog`.

Type: string
Default: „tarantool“
Environment variable: TT_LOG_SYSLOG_IDENTITY

`log.syslog.server`
Set a location of a syslog server. This option accepts one of the following values:

- An IPv4 address. Example: `127.0.0.1:514`.
- A Unix socket path starting with `unix:`. Examples: `unix:/dev/log` on Linux or `unix:/var/run/syslog` on macOS.

To write logs to syslog, you need to set *log.to* to `syslog`.

**Example**

In the example below, Tarantool writes logs to a syslog server that listens for logging messages on the `127.0.0.1:514` address:

```
log:
  to: syslog
  syslog:
    server: '127.0.0.1:514'
```

Example on GitHub: log_syslog.

Type: string
Default: box.NULL
Environment variable: TT_LOG_SYSLOG_SERVER

### 9.1.9 memtx

The `memtx` section is used to configure parameters related to the *memtx engine*.

---

**Примечание:** `memtx` can be defined in any *scope*.

---

- *memtx.allocator*
- *memtx.max_tuple_size*
- *memtx.memory*
- *memtx.min_tuple_size*
- *memtx.slab_alloc_factor*
- *memtx.slab_alloc_granularity*
- *memtx.sort_threads*

`memtx.allocator`

Specify the allocator that manages memory for `memtx` tuples. Possible values:

- `system` – the memory is allocated as needed, checking that the quota is not exceeded. THe allocator is based on the `malloc` function.
- `small` – a slab allocator. The allocator repeatedly uses a memory block to allocate objects of the same type. Note that this allocator is prone to unresolvable fragmentation on specific workloads, so you can switch to `system` in such cases.

Type: string
Default: „small“
Environment variable: TT_MEMTX_ALLOCATOR

`memtx.max_tuple_size`

Size of the largest allocation unit for the memtx storage engine in bytes. It can be increased if it is necessary to store large tuples.

Type: integer
Default: 1048576
Environment variable: TT_MEMTX_MAX_TUPLE_SIZE

`memtx.memory`

The amount of memory in bytes that Tarantool allocates to store tuples. When the limit is reached, *INSERT* and *UPDATE* requests fail with the `ER_MEMORY_ISSUE` error. The server does not go beyond the `memtx.memory` limit to allocate tuples, but there is additional memory used to store indexes and connection information.

**Example**

In the example below, the memory size is set to 1 GB (1073741824 bytes).

```
memtx:
  memory: 1073741824
```

Type: integer
Default: 268435456
Environment variable: TT_MEMTX_MEMORY

`memtx.min_tuple_size`
Size of the smallest allocation unit in bytes. It can be decreased if most of the tuples are very small.

Type: integer
Default: 16
Possible values: between 8 and 1048280 inclusive
Environment variable: TT_MEMTX_MIN_TUPLE_SIZE

`memtx.slab_alloc_factor`
The multiplier for computing the sizes of memory chunks that tuples are stored in. A lower value may result in less wasted memory depending on the total amount of memory available and the distribution of item sizes.

See also: *memtx.slab_alloc_granularity*

Type: number
Default: 1.05
Possible values: between 1 and 2 inclusive
Environment variable: TT_MEMTX_SLAB_ALLOC_FACTOR

`memtx.slab_alloc_granularity`
Specify the granularity in bytes of memory allocation in the *small allocator*. The `memtx.slab_alloc_granularity` value should meet the following conditions:

- The value is a power of two.
- The value is greater than or equal to 4.

Below are few recommendations on how to adjust the `memtx.slab_alloc_granularity` option:

- If the tuples in space are small and have about the same size, set the option to 4 bytes to save memory.
- If the tuples are different-sized, increase the option value to allocate tuples from the same `mempool` (memory pool).

See also: *memtx.slab_alloc_factor*

Type: integer
Default: 8
Environment variable: TT_MEMTX_SLAB_ALLOC_GRANULARITY

`memtx.sort_threads`
Since: *3.0.0*.

The number of threads from the *thread pool* used to sort keys of secondary indexes on loading a `memtx` database. The minimum value is 1, the maximum value is 256. The default is to use all available cores.

---

**Примечание:** Since *3.0.0*, this option replaces the approach when OpenMP threads are used to parallelize sorting. For backward compatibility, the `OMP_NUM_THREADS` environment variable is taken into account to set the number of sorting threads.

---

Type: integer
Default: box.NULL
Environment variable: TT_MEMTX_SORT_THREADS

### 9.1.10 process

The `process` defines configuration parameters of the Tarantool process in the system.

- *process.background*
- *process.coredump*
- *process.title*
- *process.pid_file*
- *process.strip_core*
- *process.username*
- *process.work_dir*

`process.background`
Run the server as a background task.

---

**Важно:** Do not enable the background mode for applications intended to run by the `tt` utility. For more information, see the *tt start* reference.

---

Type: boolean
Default: false
Environment variable: TT_PROCESS_BACKGROUND

`process.coredump`

Type: boolean
Default: false
Environment variable: TT_PROCESS_COREDUMP

`process.title`
Sets the name of the server's process title (what's shown in the COMMAND column for `ps -ef` and `top -c` commands).

Type: string
Default: „tarantool - {{ instance_name }}“
Environment variable: TT_PROCESS_TITLE

`process.pid_file`
Store the process id in this file.

Type: string
Default: „var/run/{{ instance_name }}/tarantool.pid“
Environment variable: TT_PROCESS_PID_FILE

`process.strip_core`
Whether coredump files should include memory allocated for tuples. (This can be large if Tarantool runs under heavy load.) Setting to `true` means «do not include».

Type: boolean
Default: true
Environment variable: TT_PROCESS_STRIP_CORE

`process.username`
The name of the system user to switch to after start.

Type: string
Default: box.NULL
Environment variable: TT_PROCESS_USERNAME

`process.work_dir`
A directory where Tarantool working files are stored.

Type: string
Default: box.NULL
Environment variable: TT_PROCESS_WORK_DIR

### 9.1.11 replication

The `replication` section defines configuration parameters related to *replication*.

- *replication.anon*
- *replication.bootstrap_strategy*
- *replication.connect_timeout*
- *replication.election_mode*
- *replication.election_timeout*
- *replication.election_fencing_mode*
- *replication.failover*
- *replication.peers*
- *replication.skip_conflict*
- *replication.sync_lag*
- *replication.sync_timeout*
- *replication.synchro_quorum*
- *replication.synchro_timeout*
- *replication.threads*
- *replication.timeout*

`replication.anon`

Type: boolean
Default: `false`
Environment variable: TT_REPLICATION_ANON

`replication.bootstrap_strategy`
Specifies a strategy used to bootstrap a *replica set*. The following strategies are available:

- `auto`: a node doesn't boot if half or more of the other nodes in a replica set are not connected. For example, if a replica set contains 2 or 3 nodes, a node requires 2 connected instances. In the case of 4 or 5 nodes, at least 3 connected instances are required. Moreover, a bootstrap leader fails to boot unless every connected node has chosen it as a bootstrap leader.
- `config`: use the specified node to bootstrap a replica set. To specify the bootstrap leader, use the *<replicaset_name>.bootstrap_leader* option.
- `supervised`: a bootstrap leader isn't chosen automatically but should be appointed using *box.ctl.make_bootstrap_leader()* on the desired node.
- `legacy` (deprecated since *2.11.0*): a node requires the *replication_connect_quorum* number of other nodes to be connected. This option is added to keep the compatibility with the current versions of Cartridge and might be removed in the future.

Type: string
Default: `auto`
Environment variable: TT_REPLICATION_BOOTSTRAP_STRATEGY

`replication.connect_timeout`
A timeout (in seconds) a replica waits when trying to connect to a master in a cluster. See *orphan status* for details.

This parameter is different from *replication.timeout*, which a master uses to disconnect a replica when the master receives no acknowledgments of heartbeat messages.

Type: number
Default: 30
Environment variable: TT_REPLICATION_CONNECT_TIMEOUT

`replication.election_mode`
A role of a replica set node in the *leader election process*.

The possible values are:

- `off`: a node doesn't participate in the election activities.
- `voter`: a node can participate in the election process but can't be a leader.
- `candidate`: a node should be able to become a leader.
- `manual`: allow to control which instance is the leader explicitly instead of relying on automated leader election. By default, the instance acts like a voter – it is read-only and may vote for other candidate instances. Once *box.ctl.promote()* is called, the instance becomes a candidate and starts a new election round. If the instance wins the elections, it becomes a leader but won't participate in any new elections.

Type: string
Default: *box.NULL* (the actual default value depends on *replication.failover*)
Environment variable: TT_REPLICATION_ELECTION_MODE

`replication.election_timeout`
Specifies the timeout (in seconds) between election rounds in the *leader election process* if the previous round ended up with a split vote.

It is quite big, and for most of the cases, it can be lowered to 300-400 ms.

To avoid the split vote repeat, the timeout is randomized on each node during every new election, from 100% to 110% of the original timeout value. For example, if the timeout is 300 ms and there are 3 nodes started the election simultaneously in the same term, they can set their election timeouts to 300, 310, and 320 respectively, or to 305, 302, and 324, and so on. In that way, the votes will never be split because the election on different nodes won't be restarted simultaneously.

Type: number
Default: 5
Environment variable: TT_REPLICATION_ELECTION_TIMEOUT

`replication.election_fencing_mode`
Specifies the *leader fencing mode* that affects the leader election process. When the parameter is set to `soft` or `strict`, the leader resigns its leadership if it has less than *replication.synchro_quorum* of alive connections to the cluster nodes. The resigning leader receives the status of a follower in the current election term and becomes read-only.

- In `soft` mode, a connection is considered dead if there are no responses for *4 * replication.timeout* seconds both on the current leader and the followers.
- In `strict` mode, a connection is considered dead if there are no responses for *2 * replication.timeout* seconds on the current leader and *4 * replication.timeout* seconds on the followers. This improves the chances that there is only one leader at any time.

Fencing applies to the instances that have the *replication.election_mode* set to `candidate` or `manual`. To turn off leader fencing, set `election_fencing_mode` to `off`.

Type: string
Default: `soft`
Possible values: `off`, `soft`, `strict`
Environment variable: TT_REPLICATION_ELECTION_FENCING_MODE

`replication.failover`
A failover mode used to take over a master role when the current master instance fails. The following modes are available:

- `off`

  Leadership in a replica set is controlled using the *database.mode* option. In this case, you can set the `database.mode` option to `rw` on all instances in a replica set to make a *master-master* configuration.

  The default `database.mode` is determined as follows: `rw` if there is one instance in a replica set; `ro` if there are several instances.

- `manual`

  Leadership in a replica set is controlled using the *<replicaset_name>.leader* option. In this case, a *master-master* configuration is forbidden.

  In the `manual` mode, the *database.mode* option cannot be set explicitly. The leader is configured in the read-write mode, all the other instances are read-only.

- `election`

  *Automated leader election* is used to control leadership in a replica set.

  In the `election` mode, *database.mode* and *<replicaset_name>.leader* shouldn't be set explicitly.

- `supervised` (Enterprise Edition only)

  Leadership in a replica set is controlled using an external failover agent.

In the `supervised` mode, *database.mode* and *<replicaset_name>.leader* shouldn't be set explicitly.

See also: *Replication tutorials*.

---

**Примечание:** `replication.failover` can be defined in the global, group, and replica set *scope*.

---

**Example**

In the example below, the following configuration options are specified:

- In the *credentials* section, the `replicator` user with the `replication` role is created.
- *iproto.advertise.peer* specifies that other instances should connect to an address defined in *iproto.listen* using the `replicator` user.
- `replication.failover` specifies that a master instance should be set manually.
- *<replicaset_name>.leader* sets `instance001` as a replica set leader.

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]

iproto:
  advertise:
    peer:
      login: replicator

replication:
  failover: manual

groups:
  group001:
    replicasets:
      replicaset001:
        leader: instance001
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          instance003:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
```

Type: string

Default: `off`
Environment variable: TT_REPLICATION_FAILOVER

`replication.peers`
URIs of instances that constitute a replica set. These URIs are used by an instance to connect to another instance as a replica.

Alternatively, you can use *iproto.advertise.peer* to specify a URI used to advertise the current instance to other cluster members.

**Example**

In the example below, the following configuration options are specified:

- In the *credentials* section, the `replicator` user with the `replication` role is created.
- `replication.peers` specifies URIs of replica set instances.

```
credentials:
  users:
    replicator:
      password: 'topsecret'
      roles: [replication]

replication:
  peers:
  - replicator:topsecret@127.0.0.1:3301
  - replicator:topsecret@127.0.0.1:3302
  - replicator:topsecret@127.0.0.1:3303
```

Type: array
Default: *box.NULL*
Environment variable: TT_REPLICATION_PEERS

`replication.skip_conflict`
By default, if a replica adds a unique key that another replica has added, replication *stops* with the `ER_TUPLE_FOUND` *error*. If `replication.skip_conflict` is set to `true`, such errors are ignored.

---

**Примечание:** Instead of saving the broken transaction to the write-ahead log, it is written as `NOP` (No operation).

---

Type: boolean
Default: false
Environment variable: TT_REPLICATION_SKIP_CONFLICT

`replication.sync_lag`
The maximum delay (in seconds) between the time when data is written to the master and the time when it is written to a replica. If `replication.sync_lag` is set to `nil` or 365 * 100 * 86400 (`TIMEOUT_INFINITY`), a replica is always considered to be «synced».

**Примечание:** This parameter is ignored during bootstrap. See *orphan status* for details.

Type: number
Default: 10
Environment variable: TT_REPLICATION_SYNC_LAG

`replication.sync_timeout`
The timeout (in seconds) that a node waits when trying to sync with other nodes in a replica set after connecting or during a *configuration update*. This could fail indefinitely if *replication.sync_lag* is smaller than network latency, or if the replica cannot keep pace with master updates. If `replication.sync_timeout` expires, the replica enters *orphan status*.

Type: number
Default: 0
Environment variable: TT_REPLICATION_SYNC_TIMEOUT

`replication.synchro_quorum`
A number of replicas that should confirm the receipt of a *synchronous* transaction before it can finish its commit.

This option supports dynamic evaluation of the quorum number. For example, the default value is `N / 2 + 1` where `N` is the current number of replicas registered in a cluster. Once any replicas are added or removed, the expression is re-evaluated automatically.

Note that the default value (`at least 50% of the cluster size + 1`) guarantees data reliability. Using a value less than the canonical one might lead to unexpected results, including a *split-brain*.

`replication.synchro_quorum` is not used on replicas. If the master fails, the pending synchronous transactions will be kept waiting on the replicas until a new master is elected.

**Примечание:** `replication.synchro_quorum` does not account for anonymous replicas.

Type: string, number
Default: `N / 2 + 1`
Environment variable: TT_REPLICATION_SYNCHRO_QUORUM

`replication.synchro_timeout`
For *synchronous replication* only. Specify how many seconds to wait for a synchronous transaction quorum replication until it is declared failed and is rolled back.

It is not used on replicas, so if the master fails, the pending synchronous transactions will be kept waiting on the replicas until a new master is elected.

Type: number
Default: 5
Environment variable: TT_REPLICATION_SYNCHRO_TIMEOUT

`replication.threads`
The number of threads spawned to decode the incoming replication data.

In most cases, one thread is enough for all incoming data. Possible values range from 1 to 1000. If there are multiple replication threads, connections to serve are distributed evenly between the threads.

Type: integer
Default: 1
Environment variable: TT_REPLICATION_THREADS

`replication.timeout`
A time interval (in seconds) used by a master to send heartbeat requests to a replica when there are no updates to send to this replica. For each request, a replica should return a heartbeat acknowledgment.

If a master or replica gets no heartbeat message for `4 * replication.timeout` seconds, a connection is dropped and a replica tries to reconnect to the master.

See also: *Monitoring a replica set*.

Type: number
Default: 1
Environment variable: TT_REPLICATION_TIMEOUT

### 9.1.12 roles

This section describes configuration parameters related to roles.

---

**Примечание:** Configuration parameters related to roles can be defined in any *scope*.

---

- *roles*
- *roles_cfg*

`roles`
**Since:** *3.0.0*.

Type: array
Default: nil
Environment variable: TT_ROLES

`roles_cfg`
**Since:** *3.0.0*.

Type: map
Default: nil
Environment variable: TT_ROLES_CFG

### 9.1.13 security

**Enterprise Edition**

Configuring security parameters is available in the Enterprise Edition only.

The `security` section defines configuration parameters related to various security settings.

**Примечание:** `security` can be defined in any *scope*.

- *security.auth_delay*
- *security.auth_retries*
- *security.auth_type*
- *security.disable_guest*
- *security.password_enforce_digits*
- *security.password_enforce_lowercase*
- *security.password_enforce_specialchars*
- *security.password_enforce_uppercase*
- *security.password_history_length*
- *security.password_lifetime_days*
- *security.password_min_length*
- *security.secure_erasing*

`security.auth_delay`
Specify a period of time (in seconds) that a specific user should wait for the next attempt after failed authentication.

The *security.auth_retries* option lets a client try to authenticate the specified number of times before `security.auth_delay` is enforced.

In the configuration below, Tarantool lets a client try to authenticate with the same username three times. At the fourth attempt, the authentication delay configured with `security.auth_delay` is enforced. This means that a client should wait 10 seconds after the first failed attempt.

```
security:
  auth_delay: 10
  auth_retries: 2
```

Type: number

Default: 0

Environment variable: TT_SECURITY_AUTH_DELAY

`security.auth_retries`

Specify the maximum number of authentication retries allowed before *security.auth_delay* is enforced. The default value is 0, which means `security.auth_delay` is enforced after the first failed authentication attempt.

The retry counter is reset after `security.auth_delay` seconds since the first failed attempt. For example, if a client tries to authenticate fewer than `security.auth_retries` times within `security.auth_delay` seconds, no authentication delay is enforced. The retry counter is also reset after any successful authentication attempt.

Type: integer

Default: 0

Environment variable: TT_SECURITY_AUTH_RETRIES

`security.auth_type`

Specify a protocol used to authenticate users. The possible values are:

- `chap-sha1`: use the CHAP protocol with `SHA-1` hashing applied to *passwords*.
- `pap-sha256`: use PAP authentication with the `SHA256` hashing algorithm.

Note that CHAP stores password hashes in the `_user` space unsalted. If an attacker gains access to the database, they may crack a password, for example, using a rainbow table. For PAP, a password is salted with a user-unique salt before saving it in the database, which keeps the database protected from cracking using a rainbow table.

To enable PAP, specify the `security.auth_type` option as follows:

```
security:
  auth_type: 'pap-sha256'
```

Type: string

Default: „chap-sha1“

Environment variable: TT_SECURITY_AUTH_TYPE

`security.disable_guest`

If **true**, turn off access over remote connections from unauthenticated or *guest* users. This option affects connections between cluster members and *net.box* connections.

Type: boolean

Default: false
Environment variable: TT_SECURITY_DISABLE_GUEST

`security.password_enforce_digits`
If **true**, a password should contain digits (0-9).

Type: boolean
Default: false
Environment variable: TT_SECURITY_PASSWORD_ENFORCE_DIGITS

`security.password_enforce_lowercase`
If **true**, a password should contain lowercase letters (a-z).

Type: boolean
Default: false
Environment variable: TT_SECURITY_PASSWORD_ENFORCE_LOWERCASE

`security.password_enforce_specialchars`
If **true**, a password should contain at least one special character (such as `&|?!@$`).

Type: boolean
Default: false
Environment variable: TT_SECURITY_PASSWORD_ENFORCE_SPECIALCHARS

`security.password_enforce_uppercase`
If **true**, a password should contain uppercase letters (A-Z).

Type: boolean
Default: false
Environment variable: TT_SECURITY_PASSWORD_ENFORCE_UPPERCASE

`security.password_history_length`
Specify the number of unique new user passwords before an old password can be reused.

---

**Примечание:** Tarantool uses the `auth_history` field in the *box.space._user* system space to store user passwords.

---

Type: integer
Default: 0
Environment variable: TT_SECURITY_PASSWORD_HISTORY_LENGTH

`security.password_lifetime_days`
Specify the maximum period of time (in days) a user can use the same password. When this period ends, a user gets the «Password expired» error on a login attempt. To restore access for such users, use *box.schema.user.passwd*.

---

**Примечание:** The default 0 value means that a password never expires.

---

Type: integer
Default: 0
Environment variable: TT_SECURITY_PASSWORD_LIFETIME_DAYS

`security.password_min_length`
Specify the minimum number of characters for a password.

Type: integer
Default: 0
Environment variable: TT_SECURITY_PASSWORD_MIN_LENGTH

`security.secure_erasing`
If **true**, forces Tarantool to overwrite a data file a few times before deletion to render recovery of a deleted file impossible. The option applies to both `.xlog` and `.snap` files as well as Vinyl data files.

Type: boolean
Default: false
Environment variable: TT_SECURITY_SECURE_ERASING

### 9.1.14 sharding

The `sharding` section defines configuration parameters related to *sharding*.

---

**Примечание:** Sharding support requires installing the *vshard* module. The minimum required version of `vshard` is 0.1.25.

---

- *sharding.bucket_count*
- *sharding.discovery_mode*
- *sharding.failover_ping_timeout*
- *sharding.lock*
- *sharding.rebalancer_disbalance_threshold*
- *sharding.rebalancer_max_receiving*
- *sharding.rebalancer_max_sending*
- *sharding.roles*
- *sharding.sched_move_quota*
- *sharding.sched_ref_quota*
- *sharding.shard_index*
- *sharding.sync_timeout*
- *sharding.zone*

`sharding.bucket_count`

The total number of buckets in a cluster.

`sharding.bucket_count` should be several orders of magnitude larger than the potential number of cluster nodes, considering potential scaling out in the future.

If the estimated number of nodes in a cluster is M, then the data set should be divided into 100M or even 1000M buckets, depending on the planned scaling out. This number is greater than the potential number of cluster nodes in the system being designed.

Keep in mind that too many buckets can cause a need to allocate more memory to store routing information. On the other hand, an insufficient number of buckets can lead to decreased granularity when *rebalancing*.

---

**Примечание:** This option should be defined at the *global level*.

---

**Example:**

```
sharding:
  bucket_count: 1000
```

Type: integer
Default: 3000
Environment variable: TT_SHARDING_BUCKET_COUNT

`sharding.discovery_mode`
: A mode of the background discovery fiber used by the router to find buckets. Learn more in *vshard.router.discovery_set()*.

---

**Примечание:** This option should be defined at the *global level*.

---

Type: string

Default: „on“

Possible values: „on“, „off“, „once“

Environment variable: TT_SHARDING_DISCOVERY_MODE

`sharding.failover_ping_timeout`
: The timeout (in seconds) after which a node is considered unavailable if there are no responses during this period. The *failover fiber* is used to detect if a node is down.

---

**Примечание:** This option should be defined at the *global level*.

---

Type: number

Default: 5

Environment variable: TT_SHARDING_FAILOVER_PING_TIMEOUT

`sharding.lock`
: Whether a replica set is *locked*. A locked replica set cannot receive new buckets nor migrate its own buckets.

---

**Примечание:** `sharding.lock` can be specified at the *replica set level* or higher.

---

Type: boolean

Default: nil

Environment variable: TT_SHARDING_LOCK

`sharding.rebalancer_disbalance_threshold`
: The maximum bucket *disbalance* threshold (in percent). The disbalance is calculated for each replica set using the following formula:

```
|etalon_bucket_count - real_bucket_count| / etalon_bucket_count * 100
```

**Примечание:** This option should be defined at the *global level*.

Type: number
Default: 1
Environment variable: TT_SHARDING_REBALANCER_DISBALANCE_THRESHOLD

`sharding.rebalancer_max_receiving`
The maximum number of buckets that can be *received in parallel* by a single replica set. This number must be limited because the rebalancer sends a large number of buckets from the existing replica sets to the newly added one. This produces a heavy load on the new replica set.

**Примечание:** This option should be defined at the *global level*.

**Example:**

Suppose, `rebalancer_max_receiving` is equal to 100 and `bucket_count` is equal to 1000. There are 3 replica sets with 333, 333, and 334 buckets on each respectively. When a new replica set is added, each replica set's `etalon_bucket_count` becomes equal to 250. Rather than receiving all 250 buckets at once, the new replica set receives 100, 100, and 50 buckets sequentially.

Type: integer
Default: 100
Environment variable: TT_SHARDING_REBALANCER_MAX_RECEIVING

`sharding.rebalancer_max_sending`
The degree of parallelism for *parallel rebalancing*.

**Примечание:** This option should be defined at the *global level*.

Type: integer
Default: 1
Maximum: 15
Environment variable: TT_SHARDING_REBALANCER_MAX_SENDING

`sharding.roles`
Roles of a replica set in regard to sharding. A replica set can have the following roles:

- `router`: a replica set acts as a *router*.
- `storage`: a replica set acts as a *storage*.

- `rebalancer`: a replica set acts as a *rebalancer*.

The `rebalancer` role is optional. If it is not specified, a rebalancer is selected automatically from master instances of replica sets.

There can be at most one replica set with the `rebalancer` role. Additionally, this replica set should have a `storage` role.

**Example:**

```
replicasets:
  storage-a:
    sharding:
      roles: [storage, rebalancer]
```

---

**Примечание:** `sharding.roles` can be specified at the *replica set level* or higher.

---

Type: array
Default: nil
Environment variable: TT_SHARDING_ROLES

`sharding.sched_move_quota`
A scheduler's bucket move quota used by the *rebalancer*.

`sched_move_quota` defines how many bucket moves can be done in a row if there are pending storage refs. Then, bucket moves are blocked and a router continues making map-reduce requests.

See also: *sharding.sched_ref_quota*.

---

**Примечание:** This option should be defined at the *global level*.

---

Type: number
Default: 1
Environment variable: TT_SHARDING_SCHED_MOVE_QUOTA

`sharding.sched_ref_quota`
A scheduler's storage ref quota used by a *router*'s map-reduce API. For example, the *vshard.router.map_callrw()* function implements consistent map-reduce over the entire cluster.

`sched_ref_quota` defines how many storage refs, therefore map-reduce requests, can be executed on the storage in a row if there are pending bucket moves. Then, storage refs are blocked and the rebalancer continues bucket moves.

See also: *sharding.sched_move_quota*.

---

**Примечание:** This option should be defined at the *global level*.

---

Type: number
Default: 300
Environment variable: TT_SHARDING_SCHED_REF_QUOTA

`sharding.shard_index`
The name or ID of a TREE index over the *bucket id*. Spaces without this index do not participate in a sharded Tarantool cluster and can be used as regular spaces if needed. It is necessary to specify the first part of the index, other parts are optional.

---

**Примечание:** This option should be defined at the *global level*.

---

Type: string
Default: „bucket_id“
Environment variable: TT_SHARDING_SHARD_INDEX

`sharding.sync_timeout`
The timeout to wait for synchronization of the old master with replicas before demotion. Used when switching a master or when manually calling the *sync()* function.

---

**Примечание:** This option should be defined at the *global level*.

---

Type: number
Default: 1
Environment variable: TT_SHARDING_SYNC_TIMEOUT

`sharding.zone`
A *zone* that can be set for routers and replicas. This allows sending read-only requests not only to a master instance but to any available replica that is the nearest to the router.

---

**Примечание:** `sharding.zone` can be specified at any *level*.

---

Type: integer
Default: nil
Environment variable: TT_SHARDING_ZONE

### 9.1.15 snapshot

The `snapshot` section defines configuration parameters related to the *snapshot files*. To learn more about the snapshots“ configuration, check the *Persistence* page.

---

**Примечание:** `snapshot` can be defined in any *scope*.

---

- *snapshot.dir*
- *snapshot.snap_io_rate_limit*
- *snapshot.count*
- *snapshot.by.**
  - *snapshot.by.interval*
  - *snapshot.by.wal_size*

`snapshot.dir`
: A directory where memtx stores snapshot (.snap) files. A relative path in this option is interpreted as relative to `process.work_dir`.

  By default, snapshots and WAL files are stored in the same directory. However, you can set different values for the `snapshot.dir` and *wal.dir* options to store them on different physical disks for performance matters.

  Type: string
  Default: „var/lib/{{ instance_name }}“
  Environment variable: TT_SNAPSHOT_DIR

`snapshot.snap_io_rate_limit`
: Reduce the throttling effect of *box.snapshot()* on INSERT/UPDATE/DELETE performance by setting a limit on how many megabytes per second it can write to disk. The same can be achieved by splitting *wal.dir* and *snapshot.dir* locations and moving snapshots to a separate disk. The limit also affects what *box.stat.vinyl().regulator* may show for the write rate of dumps to `.run` and `.index` files.

  Type: number
  Default: box.NULL
  Environment variable: TT_SNAPSHOT_SNAP_IO_RATE_LIMIT

`snapshot.count`
: The maximum number of snapshots that are stored in the *snapshot.dir* directory. If the number of snapshots after creating a new one exceeds this value, the *Tarantool garbage collector* deletes old snapshots. If `snapshot.count` is set to zero, the garbage collector does not delete old snapshots.

  **Example**

In the example, the checkpoint daemon creates a snapshot every two hours until it has created three snapshots. After creating a new snapshot (the fourth one), the oldest snapshot and any associated write-ahead-log files are deleted.

```
snapshot:
  by:
    interval: 7200
  count: 3
```

**Примечание:** Snapshots will not be deleted if replication is ongoing and the file has not been relayed to a replica. Therefore, `snapshot.count` has no effect unless all replicas are alive.

Type: integer
Default: 2
Environment variable: TT_SNAPSHOT_COUNT

### snapshot.by.*

`snapshot.by.interval`
The interval in seconds between actions by the *checkpoint daemon*. If the option is set to a value greater than zero, and there is activity that causes change to a database, then the checkpoint daemon calls *box.snapshot()* every `snapshot.by.interval` seconds, creating a new snapshot file each time. If the option is set to zero, the checkpoint daemon is disabled.

**Example**

In the example, the checkpoint daemon creates a new database snapshot every two hours, if there is activity.

```
by:
  interval: 7200
```

Type: number
Default: 3600
Environment variable: TT_SNAPSHOT_BY_INTERVAL

`snapshot.by.wal_size`
The threshold for the total size in bytes for all WAL files created since the last snapshot taken. Once the configured threshold is exceeded, the WAL thread notifies the *checkpoint daemon* that it must make a new snapshot and delete old WAL files.

Type: integer
Default: 10^18
Environment variable: TT_SNAPSHOT_BY_WAL_SIZE

### 9.1.16 wal

The `wal` section defines configuration parameters related to *write-ahead log*. To learn more about the WAL configuration, check the *Persistence* page.

---

**Примечание:** `wal` can be defined in any *scope*.

---

- *wal.cleanup_delay*
- *wal.dir*
- *wal.dir_rescan_delay*
- *wal.max_size*
- *wal.mode*
- *wal.queue_max_size*
- *wal.ext.**
  - *wal.ext.new*
  - *wal.ext.old*
  - *wal.ext.spaces*

`wal.cleanup_delay`

The delay in seconds used to prevent the *Tarantool garbage collector* from immediately removing *write-ahead log* files after a node restart. This delay eliminates possible erroneous situations when the master deletes WALs needed by *replicas* after restart. As a consequence, replicas sync with the master faster after its restart and don't need to download all the data again. Once all the nodes in the replica set are up and running, a scheduled garbage collection is started again even if `wal.cleanup_delay` has not expired.

---

**Примечание:** The option has no effect on nodes running as *anonymous replicas*.

---

Type: number

Default: 14400

Environment variable: TT_WAL_CLEANUP_DELAY

`wal.dir`

A directory where write-ahead log (`.xlog`) files are stored. A relative path in this option is interpreted as relative to `process.work_dir`.

By default, WAL files and snapshots are stored in the same directory. However, you can set different values for the `wal.dir` and *snapshot.dir* options to store them on different physical disks for performance matters.

Type: string

Default: „var/lib/{{ instance_name }}“

Environment variable: TT_WAL_DIR

`wal.dir_rescan_delay`
The time interval in seconds between periodic scans of the write-ahead-log file directory, when checking for changes to write-ahead-log files for the sake of *replication* or *hot standby*.

Type: number
Default: 2
Environment variable: TT_WAL_DIR_RESCAN_DELAY

`wal.max_size`
The maximum number of bytes in a single write-ahead log file. When a request would cause an `.xlog` file to become larger than `wal.max_size`, Tarantool creates a new WAL file.

Type: integer
Default: 268435456
Environment variable: TT_WAL_MAX_SIZE

`wal.mode`
Specify fiber-WAL-disk synchronization mode as:

- `none`: write-ahead log is not maintained. A node with `wal.mode` set to `none` can't be a replication master.
- `write`: *fibers* wait for their data to be written to the write-ahead log (no `fsync(2)`).
- `fsync`: fibers wait for their data, *`fsync(2)`* follows each *`write(2)`*.

Type: string
Default: „write“
Environment variable: TT_WAL_MODE

`wal.queue_max_size`
The size of the queue in bytes used by a *replica* to submit new transactions to a *write-ahead log* (WAL). This option helps limit the rate at which a replica submits transactions to the WAL.

Limiting the queue size might be useful when a replica is trying to sync with a master and reads new transactions faster than writing them to the WAL.

---

**Примечание:** You might consider increasing the `wal.queue_max_size` value in case of large tuples (approximately one megabyte or larger).

---

Type: integer
Default: 16777216
Environment variable: TT_WAL_QUEUE_MAX_SIZE

### wal.ext.*

**Enterprise Edition**

Configuring `wal.ext.*` parameters is available in the Enterprise Edition only.

This section describes options related to *WAL extensions*.

`wal.ext.new`
Enable storing a new tuple for each *CRUD* operation performed. The option is in effect for all spaces. To adjust the option for specific spaces, use the *wal.ext.spaces* option.

Type: boolean
Default: false
Environment variable: TT_WAL_EXT_NEW

`wal.ext.old`
Enable storing an old tuple for each *CRUD* operation performed. The option is in effect for all spaces. To adjust the option for specific spaces, use the *wal.ext.spaces* option.

Type: boolean
Default: false
Environment variable: TT_WAL_EXT_OLD

`wal.ext.spaces`
Enable or disable storing an old and new tuple in the *WAL* record for a given space explicitly. The configuration for specific spaces has priority over the configuration in the *wal.ext.new* and *wal.ext.old* options.

The option is a key-value pair:

- The key is a space name (string).
- The value is a table that includes two optional boolean options: `old` and `new`. The format and the default value of these options are described in `wal.ext.old` and `wal.ext.new`.

**Example**

In the example, only new tuples are added to the log for the `bands` space.

```
ext:
  new: true
  old: true
  spaces:
    bands:
      old: false
```

Type: map
Default: nil
Environment variable: TT_WAL_EXT_SPACES

## 9.2 Configuration reference (box.cfg)

**Примечание:** Starting with the 3.0 version, the recommended way of configuring Tarantool is using a *configuration file*. Configuring Tarantool in code is considered a legacy approach.

This topic describes all configuration parameters that can be specified *in code* using the `box.cfg` API.


- *Базовые параметры*
- *Настройка хранения*
- *Демон создания контрольных точек*
- *Записи в бинарный журнал и создание снимков*
- *Горячее резервирование*
- *Репликация*
- *Работа с сетями*
- *Запись в журнал*
- *Audit log*
- *Authentication*
- *Flight recorder*
- *Обратная связь*
- *Устаревшие параметры*

### 9.2.1 Базовые параметры

- *background*
- *custom_proc_title*
- *listen*
- *memtx_dir*
- *pid_file*
- *read_only*
- *sql_cache_size*
- *vinyl_dir*
- *vinyl_timeout*
- *username*
- *wal_dir*
- *work_dir*
- *worker_pool_threads*
- *strip_core*
- *memtx_use_mvcc_engine*

`background`
Since version 1.6.2.

Run the server as a background task. The *log* and *pid_file* parameters must be non-null for this to work.

---

**Важно:** Do not enable the background mode for applications intended to run by the `tt` utility. For more information, see the *tt start* reference.

---

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_BACKGROUND
Динамический: нет

`custom_proc_title`
Since version 1.6.7.

Add the given string to the server's process title (what's shown in the COMMAND column for `ps -ef` and `top -c` commands).

Например, как правило, `ps -ef` показывает процесс Tarantool-сервера так:

```
$ ps -ef | grep tarantool
1000     14939 14188  1 10:53 pts/2    00:00:13 tarantool <running>
```

Но если указан конфигурационный параметр `custom_proc_title='sessions'`, вывод выглядит так:

```
$ ps -ef | grep tarantool
1000     14939 14188  1 10:53 pts/2    00:00:16 tarantool <running>: sessions
```

Тип: строка

По умолчанию: null

Environment variable: TT_CUSTOM_PROC_TITLE

Динамический: да

`listen`

Since version 1.6.4.

The read/write data port number or *URI* (Universal Resource Identifier) string. Has no default value, so **must be specified** if connections occur from the remote clients that don't use the *«admin port»*. Connections made with `listen = URI` are called «binary port» or «binary protocol» connections.

Как правило, используется значение 3301.

```
box.cfg { listen = 3301 }

box.cfg { listen = "127.0.0.1:3301" }
```

---

**Примечание:** Реплика также привязана на этот порт и принимает соединения, но эти соединения служат только для чтения до тех пор, пока реплика не станет мастером.

---

Starting from version 2.10.0, you can specify *several URIs*, and the port number is always stored as an integer value.

Тип: целое число или строка

По умолчанию: null

Environment variable: TT_LISTEN

Динамический: да

`memtx_dir`

Since version 1.7.4.

A directory where memtx stores snapshot (`.snap`) files. A relative path in this option is interpreted as relative to *work_dir*.

By default, snapshots and WAL files are stored in the same directory. However, you can set different values for the `memtx_dir` and *wal_dir* options to store them on different physical disks for performance matters.

Тип: строка
По умолчанию: «.»
Environment variable: TT_MEMTX_DIR
Динамический: нет

`pid_file`
Since version 1.4.9.

Store the process id in this file. Can be relative to *work_dir*. A typical value is "`tarantool.pid`".

Тип: строка
По умолчанию: null
Environment variable: TT_PID_FILE
Динамический: нет

`read_only`
Since version 1.7.1.

Say `box.cfg{read_only=true...}` to put the server instance in read-only mode. After this, any requests that try to change persistent data will fail with error `ER_READONLY`. Read-only mode should be used for master-replica *replication*. Read-only mode does not affect data-change requests for spaces defined as *temporary*. Although read-only mode prevents the server from writing to the *WAL*, it does not prevent writing diagnostics with the *log module*.

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_READ_ONLY
Динамический: да

Setting `read_only == true` affects spaces differently depending on the *options* that were used during *box.schema.space.create*, as summarized by this chart:

| Характеристика | Можно создать? | Допускает запись? | Реплицируется? | Сохраняется? |
|---|---|---|---|---|
| (по умолчанию) | нет | нет | да | да |
| temporary | нет | да | нет | нет |
| is_local | нет | да | нет | да |

`sql_cache_size`
Since version 2.3.1.

The maximum number of bytes in the cache for *SQL prepared statements*. (The number of bytes that are actually used can be seen with *box.info.sql().cache.size*.)

Тип: число

По умолчанию: 5242880
Environment variable: TT_SQL_CACHE_SIZE
Динамический: да

`vinyl_dir`
Since version 1.7.1.

A directory where vinyl files or subdirectories will be stored. Can be relative to *work_dir*. If not specified, defaults to `work_dir`.

Тип: строка
По умолчанию: «.»
Environment variable: TT_VINYL_DIR
Динамический: нет

`vinyl_timeout`
Since version 1.7.5.

The vinyl storage engine has a scheduler which does compaction. When vinyl is low on available memory, the compaction scheduler may be unable to keep up with incoming update requests. In that situation, queries may time out after `vinyl_timeout` seconds. This should rarely occur, since normally vinyl would throttle inserts when it is running low on compaction bandwidth. Compaction can also be ordered manually with *index_object:compact()*.

Тип: число с плавающей запятой
По умолчанию: 60
Environment variable: TT_VINYL_TIMEOUT
Динамический: да

`username`
Since version 1.4.9.

UNIX user name to switch to after start.

Тип: строка
По умолчанию: null
Environment variable: TT_USERNAME
Динамический: нет

`wal_dir`
Since version 1.6.2.

A directory where write-ahead log (`.xlog`) files are stored. A relative path in this option is interpreted as relative to *work_dir*.

By default, WAL files and snapshots are stored in the same directory. However, you can set different values for the `wal_dir` and *memtx_dir* options to store them on different physical disks for performance matters.

Тип: строка

По умолчанию: «.»

Environment variable: TT_WAL_DIR

Динамический: нет

`work_dir`

Since version 1.4.9.

A directory where database working files will be stored. The server instance switches to `work_dir` with *chdir(2)* after start. Can be relative to the current directory. If not specified, defaults to the current directory. Other directory parameters may be relative to `work_dir`, for example:

```
box.cfg{
    work_dir = '/home/user/A',
    wal_dir = 'B',
    memtx_dir = 'C'
}
```

поместит xlog-файлы в `/home/user/A/B`, файлы снимков в `/home/user/A/C`, а все остальные файлы или поддиректории в `/home/user/A`.

Тип: строка

По умолчанию: null

Environment variable: TT_WORK_DIR

Динамический: нет

`worker_pool_threads`

Since version 1.7.5.

The maximum number of threads to use during execution of certain internal processes (currently *socket.getaddrinfo()* and *coio_call()*).

Тип: целое число

По умолчанию: 4

Environment variable: TT_WORKER_POOL_THREADS

Динамический: да

`strip_core`

Since version 2.2.2.

Whether coredump files should include memory allocated for tuples. (This can be large if Tarantool runs under heavy load.) Setting to `true` means «do not include». In an older version of Tarantool the default value of this parameter was `false`.

Тип: логический
По умолчанию: true
Environment variable: TT_STRIP_CORE
Динамический: нет

`memtx_use_mvcc_engine`
Since version *2.6.1*.

Enable *transactional manager* if set to `true`.

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_MEMTX_USE_MVCC_ENGINE
Динамический: нет

### 9.2.2 Настройка хранения

- *memtx_memory*
- *memtx_max_tuple_size*
- *memtx_min_tuple_size*
- *memtx_allocator*
- *memtx_sort_threads*
- *slab_alloc_factor*
- *slab_alloc_granularity*
- *vinyl_bloom_fpr*
- *vinyl_cache*
- *vinyl_max_tuple_size*
- *vinyl_memory*
- *vinyl_page_size*
- *vinyl_range_size*
- *vinyl_run_count_per_level*
- *vinyl_run_size_ratio*
- *vinyl_read_threads*
- *vinyl_write_threads*

`memtx_memory`
Since version 1.7.4.

How much memory Tarantool allocates to store tuples. When the limit is reached, *INSERT* or *UPDATE* requests begin failing with error `ER_MEMORY_ISSUE`. The server does not go beyond the `memtx_memory` limit to allocate tuples, but there is additional memory used to store indexes and connection information.

Тип: число с плавающей запятой
По умолчанию: 256 * 1024 * 1024 = 268435456 байтов
Minimum: 33554432 bytes (32 MB)
Environment variable: TT_MEMTX_MEMORY
Dynamic: yes but it cannot be decreased

`memtx_max_tuple_size`
Since version 1.7.4.

Size of the largest allocation unit, for the memtx storage engine. It can be increased if it is necessary to store large tuples.

Тип: целое число
По умолчанию: 1024 * 1024 = 1048576 байтов
Environment variable: TT_MEMTX_MAX_TUPLE_SIZE
Динамический: да

`memtx_min_tuple_size`
Since version 1.7.4.

Size of the smallest allocation unit. It can be decreased if most of the tuples are very small.

Тип: целое число
По умолчанию: 16 байтов
Possible values: between 8 and 1048280 inclusive
Environment variable: TT_MEMTX_MIN_TUPLE_SIZE
Динамический: нет

`memtx_allocator`
Since version *2.10.0*.

Specify the allocator that manages memory for `memtx` tuples. Possible values:

- `system` – the memory is allocated as needed, checking that the quota is not exceeded. THe allocator is based on the `malloc` function.

- `small` – a slab allocator. The allocator repeatedly uses a memory block to allocate objects of the same type. Note that this allocator is prone to unresolvable fragmentation on specific workloads, so you can switch to `system` in such cases.

Тип: строка

Default: „small“

Environment variable: TT_MEMTX_ALLOCATOR

Динамический: нет

`memtx_sort_threads`

Since: *3.0.0*.

The number of threads from the *thread pool* used to sort keys of secondary indexes on loading a `memtx` database. The minimum value is 1, the maximum value is 256. The default is to use all available cores.

---

**Примечание:** Since *3.0.0*, this option replaces the approach when OpenMP threads are used to parallelize sorting. For backward compatibility, the `OMP_NUM_THREADS` environment variable is taken into account to set the number of sorting threads.

---

Тип: целое число

Default: box.NULL

Environment variable: TT_MEMTX_SORT_THREADS

Динамический: нет

`slab_alloc_factor`

The multiplier for computing the sizes of memory chunks that tuples are stored in. A lower value may result in less wasted memory depending on the total amount of memory available and the distribution of item sizes.

See also: *slab_alloc_granularity*

Тип: число с плавающей запятой

Default: 1.05

Possible values: between 1 and 2 inclusive

Environment variable: TT_SLAB_ALLOC_FACTOR

Динамический: нет

`slab_alloc_granularity`

Since version *2.8.1*.

Specify the granularity (in bytes) of memory allocation in the *small allocator*. The `memtx.slab_alloc_granularity` value should meet the following conditions:

- The value is a power of two.
- The value is greater than or equal to 4.

Below are few recommendations on how to adjust the `memtx.slab_alloc_granularity` option:

- If the tuples in space are small and have about the same size, set the option to 4 bytes to save memory.
- If the tuples are different-sized, increase the option value to allocate tuples from the same `mempool` (memory pool).

See also: *slab_alloc_factor*

Тип: число
Default: 8 bytes
Environment variable: TT_SLAB_ALLOC_GRANULARITY
Динамический: нет

`vinyl_bloom_fpr`
Since version 1.7.4.

Bloom filter false positive rate – the suitable probability of the bloom filter to give a wrong result. The `vinyl_bloom_fpr` setting is a default value for one of the options in the *Options for space_object:create_index()* chart.

Тип: число с плавающей запятой
По умолчанию: 0.05
Environment variable: TT_VINYL_BLOOM_FPR
Динамический: нет

`vinyl_cache`
Для версий от 1.7.4. и выше. Размер кэша для движка базы данных vinyl. Размер кэша можно изменить динамически.

Тип: целое число
По умолчанию: 128 * 1024 * 1024 = 134217728 байтов
Environment variable: TT_VINYL_CACHE
Динамический: да

`vinyl_max_tuple_size`
Since version 1.7.5.

Size of the largest allocation unit, for the vinyl storage engine. It can be increased if it is necessary to store large tuples. See also: *memtx_max_tuple_size*.

Тип: целое число
По умолчанию: 1024 * 1024 = 1048576 байтов
Environment variable: TT_VINYL_MAX_TUPLE_SIZE
Динамический: нет

`vinyl_memory`
Since version 1.7.4.

The maximum number of in-memory bytes that vinyl uses.

Тип: целое число
По умолчанию: 128 * 1024 * 1024 = 134217728 байтов
Environment variable: TT_VINYL_MEMORY
Dynamic: yes but it cannot be decreased

`vinyl_page_size`
Since version 1.7.4.

Page size. Page is a read/write unit for vinyl disk operations. The `vinyl_page_size` setting is a default value for one of the options in the *Options for space_object:create_index()* chart.

Тип: целое число
По умолчанию: 8 * 1024 = 8192 байтов
Environment variable: TT_VINYL_PAGE_SIZE
Динамический: нет

`vinyl_range_size`
Since version 1.7.4.

The default maximum range size for a vinyl index, in bytes. The maximum range size affects the decision whether to *split* a range.

Если `vinyl_range_size` содержит не нулевое значение nil и не 0, это значение используется в качестве значения по умолчанию для параметра `range_size` в таблице *Параметры space_object:create_index()*.

Если `vinyl_range_size` содержит нулевое значение nil или 0, а параметр `range_size` не задан при создании индекса, то Tarantool сам задает это значение позднее в результате оценки производительности. Чтобы узнать текущее значение, используйте *index_object:stat().range_size*.

До версии Tarantool 1.10.2 значение `vinyl_range_size` по умолчанию было 1073741824.

Тип: целое число
По умолчанию: nil
Environment variable: TT_VINYL_RANGE_SIZE

Динамический: нет

`vinyl_run_count_per_level`
Since version 1.7.4.

The maximal number of runs per level in vinyl LSM tree. If this number is exceeded, a new level is created. The `vinyl_run_count_per_level` setting is a default value for one of the options in the *Options for space_object:create_index()* chart.

Тип: целое число
По умолчанию: 2
Environment variable: TT_VINYL_RUN_COUNT_PER_LEVEL
Динамический: нет

`vinyl_run_size_ratio`
Since version 1.7.4.

Ratio between the sizes of different levels in the LSM tree. The `vinyl_run_size_ratio` setting is a default value for one of the options in the *Options for space_object:create_index()* chart.

Тип: число с плавающей запятой
По умолчанию: 3.5
Environment variable: TT_VINYL_RUN_SIZE_RATIO
Динамический: нет

`vinyl_read_threads`
Since version 1.7.5.

The maximum number of read threads that vinyl can use for some concurrent operations, such as I/O and compression.

Тип: целое число
По умолчанию: 1
Environment variable: TT_VINYL_READ_THREADS
Динамический: нет

`vinyl_write_threads`
Since version 1.7.5.

The maximum number of write threads that vinyl can use for some concurrent operations, such as I/O and compression.

Тип: целое число
По умолчанию: 4
Environment variable: TT_VINYL_WRITE_THREADS
Динамический: нет

### 9.2.3 Демон создания контрольных точек

- *checkpoint_count*
- *checkpoint_interval*
- *checkpoint_wal_threshold*

**Checkpoint daemon**

The checkpoint daemon (snapshot daemon) is a constantly running *fiber*. The checkpoint daemon creates a schedule for the periodic snapshot creation based on the configuration options and the speed of file size growth. If enabled, the daemon makes new *snapshot* (`.snap`) files according to this schedule.

The work of the checkpoint daemon is based on the following configuration options:

- *checkpoint_interval* – a new snapshot is taken once in a given period.
- *checkpoint_wal_threshold* – a new snapshot is taken once the size of all WAL files created since the last snapshot exceeds a given limit.

If necessary, the checkpoint daemon also activates the *Tarantool garbage collector* that deletes old snapshots and WAL files.

**Сборщик мусора Tarantool**

Tarantool garbage collector can be activated by the *checkpoint daemon*. The garbage collector tracks the snapshots that are to be *relayed to a replica* or needed by other consumers. When the files are no longer needed, Tarantool garbage collector deletes them.

---

**Примечание:** The garbage collector called by the checkpoint daemon is distinct from the Lua garbage collector which is for Lua objects, and distinct from the Tarantool garbage collector that specializes in *handling shard buckets*.

---

This garbage collector is called as follows:

- When the number of snapshots reaches the limit of *checkpoint_count* size. After a new snapshot is taken, Tarantool garbage collector deletes the oldest snapshot file and any associated WAL files.
- When the size of all WAL files created since the last snapshot reaches the limit of *checkpoint_wal_threshold*. Once this size is exceeded, the checkpoint daemon takes a snapshot, then the garbage collector deletes the old WAL files.

If an old snapshot file is deleted, the Tarantool garbage collector also deletes any *write-ahead log (.xlog)* files that meet the following conditions:

- The WAL files are older than the snapshot file.
- The WAL files contain information present in the snapshot file.

Tarantool garbage collector also deletes obsolete vinyl `.run` files.

Tarantool garbage collector doesn't delete a file in the following cases:

- A backup is running, and the file has not been backed up (see *Hot backup*).

- Replication is running, and the file has not been relayed to a replica (see *Replication architecture*),
- A replica is connecting.
- A replica has fallen behind. The progress of each replica is tracked; if a replica's position is far from being up to date, then the server stops to give it a chance to catch up. If an administrator concludes that a replica is permanently down, then the correct procedure is to restart the server, or (preferably) *remove the replica from the cluster*.

`checkpoint_interval`
Since version 1.7.4.

The interval in seconds between actions by the *checkpoint daemon*. If the option is set to a value greater than zero, and there is activity that causes change to a database, then the checkpoint daemon calls *box.snapshot()* every `checkpoint_interval` seconds, creating a new snapshot file each time. If the option is set to zero, the checkpoint daemon is disabled.

**Example**

```
box.cfg{ checkpoint_interval = 7200 }
```

In the example, the checkpoint daemon creates a new database snapshot every two hours, if there is activity.

Тип: целое число
По умолчанию: 3600 (один час)
Environment variable: TT_CHECKPOINT_INTERVAL
Динамический: да

`checkpoint_count`
Since version 1.7.4.

The maximum number of snapshots that are stored in the *memtx_dir* directory. If the number of snapshots after creating a new one exceeds this value, the *Tarantool garbage collector* deletes old snapshots. If the option is set to zero, the garbage collector does not delete old snapshots.

**Example**

```
box.cfg{
    checkpoint_interval = 7200,
    checkpoint_count  = 3
}
```

In the example, the checkpoint daemon creates a new snapshot every two hours until it has created three snapshots. After creating a new snapshot (the fourth one), the oldest snapshot and any associated write-ahead-log files are deleted.

---

**Примечание:** Snapshots will not be deleted if replication is ongoing and the file has not been relayed to a replica. Therefore, `checkpoint_count` has no effect unless all replicas are alive.

---

Тип: целое число

По умолчанию: 2
Environment variable: TT_CHECKPOINT_COUNT
Динамический: да

`checkpoint_wal_threshold`
Since version 2.1.2.

The threshold for the total size in bytes for all WAL files created since the last checkpoint. Once the configured threshold is exceeded, the WAL thread notifies the *checkpoint daemon* that it must make a new checkpoint and delete old WAL files.

Этот параметр позволяет администраторам справиться с проблемой, которая может возникнуть при вычислении объема дискового пространства для раздела, содержащего файлы WAL.

Тип: целое число
По умолчанию: 10^18 (большое число: можно считать, что предела нет)
Environment variable: TT_CHECKPOINT_WAL_THRESHOLD
Динамический: да

### 9.2.4 Записи в бинарный журнал и создание снимков

- *force_recovery*
- *wal_max_size*
- *snap_io_rate_limit*
- *wal_mode*
- *wal_dir_rescan_delay*
- *wal_queue_max_size*
- *wal_cleanup_delay*
- *wal_ext*
- *secure_erasing*

`force_recovery`
Since version 1.7.4.

If `force_recovery` equals true, Tarantool tries to continue if there is an error while reading a *snapshot file* (at server instance start) or a *write-ahead log file* (at server instance start or when applying an update at a replica): skips invalid records, reads as much data as possible and lets the process finish with a warning. Users can prevent the error from recurring by writing to the database and executing *box.snapshot()*.

В остальных случаях Tarantool прерывает восстановление на ошибке чтения.

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_FORCE_RECOVERY

Динамический: нет

`wal_max_size`
Since version 1.7.4.

The maximum number of bytes in a single write-ahead log file. When a request would cause an `.xlog` file to become larger than `wal_max_size`, Tarantool creates a new WAL file.

Тип: целое число
По умолчанию: 268435456 (256 * 1024 * 1024) байтов
Environment variable: TT_WAL_MAX_SIZE
Динамический: нет

`snap_io_rate_limit`
Since version 1.4.9.

Reduce the throttling effect of *box.snapshot()* on INSERT/UPDATE/DELETE performance by setting a limit on how many megabytes per second it can write to disk. The same can be achieved by splitting *wal_dir* and *memtx_dir* locations and moving snapshots to a separate disk. The limit also affects what *box.stat.vinyl().regulator* may show for the write rate of dumps to `.run` and `.index` files.

Тип: число с плавающей запятой
По умолчанию: null
Environment variable: TT_SNAP_IO_RATE_LIMIT
Динамический: да

`wal_mode`
Since version 1.6.2.

Specify fiber-WAL-disk synchronization mode as:

- `none`: write-ahead log is not maintained. A node with `wal_mode` set to `none` can't be a replication master.
- `write`: *fibers* wait for their data to be written to the write-ahead log (no *fsync(2)* ).
- `fsync`: fibers wait for their data, *fsync(2)* follows each *write(2)*.

Тип: строка
По умолчанию: «write»
Environment variable: TT_WAL_MODE
Динамический: нет

`wal_dir_rescan_delay`
Since version 1.6.2.

The time interval in seconds between periodic scans of the write-ahead-log file directory, when checking for changes to write-ahead-log files for the sake of *replication* or *hot standby*.

Тип: число с плавающей запятой

По умолчанию: 2

Environment variable: TT_WAL_DIR_RESCAN_DELAY

Динамический: нет

`wal_queue_max_size`

Since version *2.8.1*.

The size of the queue (in bytes) used by a *replica* to submit new transactions to a *write-ahead log* (WAL). This option helps limit the rate at which a replica submits transactions to the WAL. Limiting the queue size might be useful when a replica is trying to sync with a master and reads new transactions faster than writing them to the WAL.

---

**Примечание:** You might consider increasing the `wal_queue_max_size` value in case of large tuples (approximately one megabyte or larger).

---

Тип: число

Default: 16777216 bytes

Environment variable: TT_WAL_QUEUE_MAX_SIZE

Динамический: да

`wal_cleanup_delay`

Since version *2.6.3*.

The delay in seconds used to prevent the *Tarantool garbage collector* from immediately removing *write-ahead log* files after a node restart. This delay eliminates possible erroneous situations when the master deletes WALs needed by *replicas* after restart. As a consequence, replicas sync with the master faster after its restart and don't need to download all the data again. Once all the nodes in the replica set are up and running, a scheduled garbage collection is started again even if `wal_cleanup_delay` has not expired.

---

**Примечание:** The `wal_cleanup_delay` option has no effect on nodes running as *anonymous replicas*.

---

Тип: число

Default: 14400 seconds

Environment variable: TT_WAL_CLEANUP_DELAY

Динамический: да

`wal_ext`

Since version *2.11.0*.

(**Enterprise Edition only**) Allows you to add auxiliary information to each *write-ahead log* record. For example, you can enable storing an old and new tuple for each CRUD operation performed. This information might be helpful for implementing a CDC (Change Data Capture) utility that transforms a data replication stream.

You can enable storing old and new tuples as follows:

- Set the `old` and `new` options to `true` to store old and new tuples in a write-ahead log for all spaces.

```
box.cfg {
    wal_ext = { old = true, new = true }
}
```

- To adjust these options for specific spaces, use the `spaces` option.

```
box.cfg {
    wal_ext = {
        old = true, new = true,
        spaces = {
            space1 = { old = false },
            space2 = { new = false }
        }
    }
}
```

The configuration for specific spaces has priority over the global configuration, so only new tuples are added to the log for `space1` and only old tuples for `space2`.

Note that records with additional fields are *replicated* as follows:

- If a replica doesn't support the extended format configured on a master, auxiliary fields are skipped.
- If a replica and master have different configurations for WAL records, the master's configuration is ignored.

Type: map

По умолчанию: nil

Environment variable: TT_WAL_EXT

Динамический: да

`secure_erasing`

Since version *3.0.0*.

(**Enterprise Edition only**) If **true**, forces Tarantool to overwrite a data file a few times before deletion to render recovery of a deleted file impossible. The option applies to both `.xlog` and `.snap` files as well as Vinyl data files.

Тип: логический

По умолчанию: false (ложь)
Environment variable: TT_SECURE_ERASING
Динамический: да

### 9.2.5 Горячее резервирование

`hot_standby`
Since version 1.7.4.

Whether to start the server in **hot standby** mode.

Горячее резервирование – это функция, которая обеспечивает простое восстановление после отказа без *репликации*.

Предполагается, что есть два экземпляра сервера, использующих одну и ту же конфигурацию. Первый из них станет «основным» экземпляром. Тот, который запускается вторым, станет «резервным» экземпляром.

Чтобы создать резервный экземпляр, запустите второй экземпляр Tarantool-сервера на том же компьютере с теми же настройками конфигурации *box.cfg* – включая одинаковые директории и ненулевые URI – и с дополнительной настройкой конфигурации `hot_standby = true`. В ближайшее время вы увидите уведомление, которое заканчивается словами `I> Entering hot standby mode` (вход в режим горячего резервирования). Всё в порядке – это означает, что резервный экземпляр готов взять работу на себя, если основной экземпляр прекратит работу.

Резервный экземпляр начнет инициализацию и попытается заблокировать *wal_dir*, но не сможет, поскольку директория `wal_dir` заблокирована основным экземпляром. Поэтому резервный экземпляр входит в цикл, выполняя чтение журнала упреждающей записи, в который записывает данные основной экземпляр (поэтому два экземпляра всегда синхронизированы), и пытаясь произвести блокировку. Если основной экземпляр по какой-либо причине прекращает работу, блокировка снимается. В таком случае резервный экземпляр сможет заблокировать директорию на себя, подключится по адресу для *прослушивания* и станет основным экземпляром. В ближайшее время вы увидите уведомление, которое заканчивается словами `I> ready to accept requests` (готов принимать запросы).

Таким образом, если основной экземпляр прекращает работу, время простоя отсутствует.

Функция горячего резервирования не работает:

- если *wal_dir_rescan_delay = большое число* (в Mac OS и FreeBSD); на этих платформах цикл запрограммирован на повторение каждые `wal_dir_rescan_delay` секунд.
- если *wal_mode = „none“*; будет работать только при `wal_mode = 'write'` или `wal_mode = 'fsync'`.
- со спейсами, созданными на движке vinyl *engine = „vinyl“*; работает с движком memtx `engine = 'memtx'`.

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_HOT_STANDBY
Динамический: нет

### 9.2.6 Репликация

- *replication*
- *replication_anon*
- *bootstrap_leader*
- *bootstrap_strategy*
- *replication_connect_timeout*
- *replication_connect_quorum*
- *replication_skip_conflict*
- *replication_sync_lag*
- *replication_sync_timeout*
- *replication_timeout*
- *replicaset_uuid*
- *instance_uuid*
- *replication_synchro_quorum*
- *replication_synchro_timeout*
- *replication_threads*
- *election_mode*
- *election_timeout*
- *election_fencing_mode*

`replication`
Since version 1.7.4.

If `replication` is not an empty string, the instance is considered to be a Tarantool *replica*. The replica will try to connect to the master specified in `replication` with a *URI* (Universal Resource Identifier), for example:

*konstantin:secret_password@tarantool.org:3301*

Если в наборе реплик более одного источника репликации, укажите массив URI, например (замените „uri“ и „uri2“ в данном примере на рабочие URI):

```
box.cfg{ replication = { 'uri1', 'uri2' } }
```

---

**Примечание:** Starting from version 2.10.0, there is a number of other ways for specifying several URIs. See *syntax examples*.

---

If one of the URIs is «self» – that is, if one of the URIs is for the instance where `box.cfg{}` is being executed – then it is ignored. Thus, it is possible to use the same `replication` specification on multiple server instances, as shown in *these examples*.

По умолчанию, пользователем считается „guest“.

Реплика в режиме только для чтения не принимает запросы по изменению данных по порту для *прослушивания*.

Параметр `replication` является динамическим, то есть для входа в режим мастера необходимо просто присвоить параметру `replication` пустую строку и выполнить следующее:

```
box.cfg{ replication = новое-значение }
```

Тип: строка
По умолчанию: null
Environment variable: TT_REPLICATION
Динамический: да

`replication_anon`

Since version 2.3.1.

A Tarantool replica can be anonymous. This type of replica is read-only (but you still can write to temporary and replica-local spaces), and it isn't present in the `_cluster` table.

Так как анонимная реплика не зарегистрирована в таблице `_cluster`, нет никаких ограничений для количества анонимных реплик в наборе: можно создавать их сколько угодно.

Чтобы сделать реплику анонимной, передайте опцию `replication_anon=true` в `box.cfg` и установите значение `read_only` равным `true`.

Попробуем создать анонимную реплику. Предположим, что у нас есть мастер с такой настройкой:

```
box.cfg{listen=3301}
```

и создан локальный спейс «loc»:

```
box.schema.space.create('loc', {is_local=true})
box.space.loc:create_index("pk")
```

Теперь, чтобы настроить анонимную реплику, нам нужно обратиться к `box.cfg`, как обычно.

```
box.cfg{replication_anon=true, read_only=true, replication=3301}
```

Как было сказано выше, `replication_anon` может быть `true` только вместе с `read_only`. Этот экземпляр получит снимок мастера и начнет следить за его изменениями. Он не получит id, поэтому его значение останется нулевым.

```
tarantool> box.info.id
---
- 0
...
tarantool> box.info.replication
---
- 1:
    id: 1
    uuid: 3c84f8d9-e34d-4651-969c-3d0ed214c60f
    lsn: 4
    upstream:
    status: follow
    idle: 0.6912029999985
    peer:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    lag: 0.00014615058898926
...
```

Теперь можно использовать реплику. Например, можно сделать вставку в локальный спейс:

```
tarantool> for i = 1,10 do
    > box.space.loc:insert{i}
    > end
---
...
```

Заметьте, что пока экземпляр анонимный, увеличится нулевая компонента его `vclock`:

```
tarantool> box.info.vclock
---
- {0: 10, 1: 4}
...
```

А теперь давайте сделаем анонимную реплику снова обычной:

```
tarantool> box.cfg{replication_anon=false}
2019-12-13 20:34:37.423 [71329] main I> assigned id 2 to replica 6a9c2ed2-b9e1-4c57-
↪a0e8-51a46def7661
2019-12-13 20:34:37.424 [71329] main/102/interactive I> set 'replication_anon'␣
↪configuration option to false
---
...

tarantool> 2019-12-13 20:34:37.424 [71329] main/117/applier/ I> subscribed
2019-12-13 20:34:37.424 [71329] main/117/applier/ I> remote vclock {1: 5} local␣
↪vclock {0: 10, 1: 5}
2019-12-13 20:34:37.425 [71329] main/118/applierw/ C> leaving orphan mode
```

Эта реплика получила id равный 2. Мы можем снова сделать ее открытой на запись:

```
tarantool> box.cfg{read_only=false}
2019-12-13 20:35:46.392 [71329] main/102/interactive I> set 'read_only'␣
↪configuration option to false
---
...

tarantool> box.schema.space.create('test')
---
- engine: memtx
before_replace: 'function: 0x01109f9dc8'
on_replace: 'function: 0x01109f9d90'
ck_constraint: []
field_count: 0
temporary: false
index: []
is_local: false
enabled: false
name: test
```

(continues on next page)

(продолжение с предыдущей страницы)

```
id: 513
- created
...

tarantool> box.info.vclock
---
- {0: 10, 1: 5, 2: 2}
...
```

Теперь, как и ожидалось, реплика отслеживает свои изменения во 2-й компоненте vclock. С этого момента она также может стать мастером репликации.

Замечания:

- Нельзя реплицироваться от анонимной реплики.
- Чтобы вернуть анонимный экземпляр к обычному состоянию, сначала запустите его как анонимный, а потом вызовите `box.cfg{replication_anon=false}`
- Чтобы деанонимизация прошла успешно, экземпляр должен быть скопирован с какого-то экземпляра, открытого на запись, иначе он не может быть добавлен в таблицу `_cluster`.

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_REPLICATION_ANON
Динамический: да

`bootstrap_leader`
Since *3.0.0*.

A bootstrap leader for a replica set. You can pass a bootstrap leader's URI, UUID, or name.

To specify a bootstrap leader manually, you need to set *bootstrap_strategy* to `config`, for example:

```
box.cfg{
    bootstrap_strategy = 'config',
    bootstrap_leader = '127.0.0.1:3301',
    replication = {'127.0.0.1:3301'},
}
```

Тип: строка
По умолчанию: null
Environment variable: TT_BOOTSTRAP_LEADER
Динамический: да

`bootstrap_strategy`
Since *2.11.0*.

Specify a strategy used to bootstrap a *replica set*. The following strategies are available:

- `auto`: a node doesn't boot if a half or more of other nodes in a replica set are not connected. For example, if the *replication* parameter contains 2 or 3 nodes, a node requires 2 connected instances. In the case of 4 or 5 nodes, at least 3 connected instances are required. Moreover, a bootstrap leader fails to boot unless every connected node has chosen it as a bootstrap leader.
- `config`: use the specified node to bootstrap a replica set. To specify the bootstrap leader, use the *bootstrap_leader* option.
- `supervised`: a bootstrap leader isn't chosen automatically but should be appointed using *box.ctl.make_bootstrap_leader()* on the desired node.
- `legacy` (deprecated since *2.11.0*): a node requires the *replication_connect_quorum* number of other nodes to be connected. This option is added to keep the compatibility with the current versions of Cartridge and might be removed in the future.

Тип: строка

Default: auto

Environment variable: TT_BOOTSTRAP_STRATEGY

Динамический: да

`replication_connect_timeout`

Since version 1.9.0.

The number of seconds that a replica will wait when trying to connect to a master in a cluster. See *orphan status* for details.

This parameter is different from *replication_timeout*, which a master uses to disconnect a replica when the master receives no acknowledgments of heartbeat messages.

Тип: число с плавающей запятой

По умолчанию: 30

Environment variable: TT_REPLICATION_CONNECT_TIMEOUT

Динамический: да

`replication_connect_quorum`

Deprecated since *2.11.0*.

This option is in effect if *bootstrap_strategy* is set to `legacy`.

Specify the number of nodes to be up and running to start a replica set. This parameter has effect during *bootstrap* or *configuration update*. Setting `replication_connect_quorum` to `0` makes Tarantool require no immediate reconnect only in case of recovery. See *Orphan status* for details.

Пример:

```
box.cfg { replication_connect_quorum = 2 }
```

Тип: целое число

По умолчанию: null

Environment variable: TT_REPLICATION_CONNECT_QUORUM

Динамический: да

`replication_skip_conflict`

Since version 1.10.1.

By default, if a replica adds a unique key that another replica has added, replication *stops* with error = ER_TUPLE_FOUND.

Однако если указать `replication_skip_conflict = true`, пользователи могут задать пропуск таких ошибок. Так, вместо сохранения сломанной транзакции в xlog, там будет записано `NOP` (No operation).

Пример:

```
box.cfg{replication_skip_conflict=true}
```

Тип: логический

По умолчанию: false (ложь)

Environment variable: TT_REPLICATION_SKIP_CONFLICT

Динамический: да

---

**Примечание:** `replication_skip_conflict = true` рекомендуется использовать только для ручного восстановления репликации.

---

`replication_sync_lag`

Since version 1.9.0.

The maximum *lag* allowed for a replica. When a replica *syncs* (gets updates from a master), it may not catch up completely. The number of seconds that the replica is behind the master is called the «lag». Syncing is considered to be complete when the replica's lag is less than or equal to `replication_sync_lag`.

Если пользователь задает значение `replication_sync_lag`, равное nil или 365 * 100 * 86400 (TIMEOUT_INFINITY), то отставание не имеет значения – реплика всегда будет синхронизирована. Кроме того, отставание не учитывается (считается бесконечным), если мастер работает на версии Tarantool старше 1.7.7, которая не отправляет *сообщения контрольного сигнала*.

Этот параметр не учитывается во время настройки. Для получения подробной информации, см. *статус orphan*.

Тип: число с плавающей запятой

По умолчанию: 10

Environment variable: TT_REPLICATION_SYNC_LAG

Динамический: да

`replication_sync_timeout`
Since version 1.10.2.

The number of seconds that a node waits when trying to sync with other nodes in a replica set (see *bootstrap_strategy*), after connecting or during *configuration update*. This could fail indefinitely if `replication_sync_lag` is smaller than network latency, or if the replica cannot keep pace with master updates. If `replication_sync_timeout` expires, the replica enters *orphan status*.

Тип: число с плавающей запятой
По умолчанию: 300
Environment variable: TT_REPLICATION_SYNC_TIMEOUT
Динамический: да

---

**Примечание:** The default `replication_sync_timeout` value is going to be changed in future versions from `300` to `0`. You can learn the reasoning behind this decision from the *Default value for replication_sync_timeout* topic, which also describes how to try the new behavior in the current version.

---

`replication_timeout`
Since version 1.7.5.

If the master has no updates to send to the replicas, it sends heartbeat messages every `replication_timeout` seconds, and each replica sends an ACK packet back.

И мастер, и реплики запрограммированы разорвать соединение при отсутствии сообщений в течение четырех промежутков времени, указанного в параметре `replication_timeout`. После разрыва соединения реплика пытается снова подключиться к мастеру.

См. дополнительную информацию в разделе *Мониторинг набора реплик*.

Тип: целое число
По умолчанию: 1
Environment variable: TT_REPLICATION_TIMEOUT
Динамический: да

`replicaset_uuid`
Since version 1.9.0.

As described in section *«Replication architecture»*, each replica set is identified by a universally unique identifier called **replica set UUID**, and each instance is identified by an **instance UUID**.

Как правило, достаточно позволить системе сгенерировать и форматировать строки, содержащие UUID, которые будут храниться постоянно.

Однако, некоторые администраторы предпочитают сохранять конфигурацию Tarantool в центральном репозитории, например, Apache ZooKeeper. Они могут самостоятельно присвоить значения экземплярам (*instance_uuid*) и набору реплик (`replicaset_uuid`) при первом запуске.

Общие правила:

- Значения должны быть действительно уникальными; они не должны одновременно принадлежать другим экземплярам или наборам реплик в той же инфраструктуре.
- Значения должны использоваться постоянно, неизменно с первого запуска (первоначальные значения хранятся в *файлах снимков* и проверяются при каждом перезапуске системы).
- Значения должны соответствовать требованиям RFC 4122. Нулевой UUID не допускается.

Формат UUID включает в себя шестнадцать октетов, представленных в виде 32 шестнадцатеричных чисел (с основанием 16) в пяти группах, разделенных дефисами в форме `8-4-4-4-12` – 36 символов (32 буквенно-цифровых символа и четыре дефиса).

Пример:

```
box.cfg{replicaset_uuid='7b853d13-508b-4b8e-82e6-806f088ea6e9'}
```

Тип: строка
По умолчанию: null
Environment variable: TT_REPLICASET_UUID
Динамический: нет

`instance_uuid`
Since version 1.9.0.

For replication administration purposes, it is possible to set the universally unique identifiers of the instance (`instance_uuid`) and the replica set (`replicaset_uuid`), instead of having the system generate the values.

Для получения подробной информации см. описание параметра *replicaset_uuid*.

Пример:

```
box.cfg{instance_uuid='037fec43-18a9-4e12-a684-a42b716fcd02'}
```

Тип: строка
По умолчанию: null
Environment variable: TT_INSTANCE_UUID
Динамический: нет

`replication_synchro_quorum`
Since version *2.5.1*.

For *synchronous replication* only. This option tells how many replicas should confirm the receipt of a synchronous transaction before it can finish its commit.

Since version *2.5.3*, the option supports dynamic evaluation of the quorum number. That is, the number of quorum can be specified not as a constant number, but as a function instead. In this case, the option returns the formula evaluated. The result is treated as an integer number. Once any replicas are added or removed, the expression is re-evaluated automatically.

For example,

```
box.cfg{replication_synchro_quorum = "N / 2 + 1"}
```

Where `N` is a current number of registered replicas in a cluster.

Keep in mind that the example above represents a canonical quorum definition. The formula `at least 50% of the cluster size + 1` guarantees data reliability. Using a value less than the canonical one might lead to unexpected results, including a *split-brain*.

Since version *2.10.0*, this option does not account for anonymous replicas.

По умолчанию значение параметра равно `N / 2 + 1`.

Она не используется на репликах, поэтому если мастер умирает, отложенные синхронные транзакции будут ждать на репликах до тех пор, пока не будет выбран новый мастер.

Если значение параметра равно 1, синхронные транзакции работают как асинхронные, пока они не настроены. 1 означает, что для коммита достаточно успешной записи в WAL на мастере.

Тип: число
Default: N / 2 + 1 (before version *2.10.0*, the default value was 1)
Environment variable: TT_REPLICATION_SYNCHRO_QUORUM
Динамический: да

`replication_synchro_timeout`

Since version *2.5.1*.

For *synchronous replication* only. Tells how many seconds to wait for a synchronous transaction quorum replication until it is declared failed and is rolled back.

Она не используется на репликах, поэтому если мастер умирает, отложенные синхронные транзакции будут ждать на репликах до тех пор, пока не будет выбран новый мастер.

Тип: число
По умолчанию: 5
Environment variable: TT_REPLICATION_SYNCHRO_TIMEOUT
Динамический: да

`replication_threads`

Since version *2.10.0*.

The number of threads spawned to decode the incoming replication data.

The default value is 1. It means that a single separate thread handles all the incoming replication streams. In most cases, one thread is enough for all incoming data. Therefore, it is likely that the user will not need to set this configuration option.

Possible values range from 1 to 1000. If there are multiple replication threads, connections to serve are distributed evenly between the threads.

Тип: число
По умолчанию: 1
Possible values: from 1 to 1000
Environment variable: TT_REPLICATION_THREADS
Dynamic: **no**

`election_mode`
Since version *2.6.1*.

Specify the role of a replica set node in the *leader election process*.

Possible values:

- off
- voter
- candidate
- manual.

Participation of a replica set node in the automated leader election can be turned on and off by this option.

The default value is `off`. All nodes that have values other than `off` run the Raft state machine internally talking to other nodes according to the Raft leader election protocol. When the option is `off`, the node accepts Raft messages from other nodes, but it doesn't participate in the election activities, and this doesn't affect the node's state. So, for example, if a node is not a leader but it has `election_mode = 'off'`, it is writable anyway.

You can control which nodes can become a leader. If you want a node to participate in the election process but don't want that it becomes a leaders, set the `election_mode` option to `voter`. In this case, the election works as usual, this particular node will vote for other nodes, but won't become a leader.

If the node should be able to become a leader, use `election_mode = 'candidate'`.

Since version *2.8.2*, the manual election mode is introduced. It may be used when a user wants to control which instance is the leader explicitly instead of relying on the Raft election algorithm.

When an instance is configured with the `election_mode='manual'`, it behaves as follows:

- By default, the instance acts like a voter – it is read-only and may vote for other instances that are candidates.
- Once *box.ctl.promote()* is called, the instance becomes a candidate and starts a new election round. If the instance wins the elections, it becomes a leader, but won't participate in any new elections.

Тип: строка
Default: „off“
Environment variable: TT_ELECTION_MODE
Динамический: да

`election_timeout`
Since version *2.6.1*.

Specify the timeout between election rounds in the *leader election process* if the previous round ended up with a split-vote.

In the *leader election process*, there can be an election timeout for the case of a split-vote. The timeout can be configured using this option; the default value is 5 seconds.

It is quite big, and for most of the cases it can be freely lowered to 300-400 ms. It can be a floating point value (300 ms would be `box.cfg{election_timeout = 0.3}`).

To avoid the split vote repeat, the timeout is randomized on each node during every new election, from 100% to 110% of the original timeout value. For example, if the timeout is 300 ms and there are 3 nodes started the election simultaneously in the same term, they can set their election timeouts to 300, 310, and 320 respectively, or to 305, 302, and 324, and so on. In that way, the votes will never be split because the election on different nodes won't be restarted simultaneously.

Тип: число

По умолчанию: 5

Environment variable: TT_ELECTION_TIMEOUT

Динамический: да

`election_fencing_mode`

Since version *2.11.0*.

In earlier Tarantool versions, use *election_fencing_enabled* instead.

Specify the *leader fencing mode* that affects the leader election process. When the parameter is set to `soft` or `strict`, the leader resigns its leadership if it has less than *replication_synchro_quorum* of alive connections to the cluster nodes. The resigning leader receives the status of a *follower* in the current election term and becomes read-only.

- In `soft` mode, a connection is considered dead if there are no responses for *4*replication_timeout* seconds both on the current leader and the followers.
- In `strict` mode, a connection is considered dead if there are no responses for *2*replication_timeout* seconds on the current leader and *4*replication_timeout* seconds on the followers. This improves chances that there is only one leader at any time.

Fencing applies to the instances that have the *election_mode* set to `candidate` or `manual`. To turn off *leader fencing*, set `election_fencing_mode` to `off`.

Тип: строка

Default: „soft“

Environment variable: TT_ELECTION_FENCING_MODE

Динамический: да

### 9.2.7 Работа с сетями

- *io_collect_interval*
- *net_msg_max*
- *readahead*
- *iproto_threads*

`io_collect_interval`
Since version 1.4.9.

The instance will sleep for io_collect_interval seconds between iterations of the event loop. Can be used to reduce CPU load in deployments in which the number of client connections is large, but requests are not so frequent (for example, each connection issues just a handful of requests per second).

Тип: число с плавающей запятой
По умолчанию: null
Environment variable: TT_IO_COLLECT_INTERVAL
Динамический: да

`net_msg_max`
Since version 1.10.1.

To handle messages, Tarantool allocates fibers. To prevent fiber overhead from affecting the whole system, Tarantool restricts how many messages the fibers handle, so that some pending requests are blocked.

В мощных системах увеличьте значение `net_msg_max`, и планировщик немедленно приступит к обработке отложенных запросов.

В более слабых системах уменьшите значение `net_msg_max`, чтобы снизить загрузку, хотя это и займет некоторое время, поскольку планировщик будет ожидать завершения уже запущенных запросов.

When `net_msg_max` is reached, Tarantool suspends processing of incoming packages until it has processed earlier messages. This is not a direct restriction of the number of fibers that handle network messages, rather it is a system-wide restriction of channel bandwidth. This in turn causes restriction of the number of incoming network messages that the *transaction processor thread* handles, and therefore indirectly affects the fibers that handle network messages. (The number of fibers is smaller than the number of messages because messages can be released as soon as they are delivered, while incoming requests might not be processed until some time after delivery.)

Для стандартных систем подойдет значение, используемое по умолчанию (768).

Тип: целое число
По умолчанию: 768
Environment variable: TT_NET_MSG_MAX
Динамический: да

`readahead`
Since version 1.6.2.

The size of the read-ahead buffer associated with a client connection. The larger the buffer, the more memory an active connection consumes and the more requests can be read from the operating system buffer in a single system call. The rule of thumb is to make sure the buffer can contain at least a few dozen requests. Therefore, if a typical tuple in a request is large, e.g. a few kilobytes or even megabytes, the read-ahead buffer size should be increased. If batched request processing is not used, it's prudent to leave this setting at its default.

Тип: целое число
По умолчанию: 16320
Environment variable: TT_READAHEAD
Динамический: да

`iproto_threads`
Since version *2.8.1*.

The number of *network threads*. There can be unusual workloads where the network thread is 100% loaded and the transaction processor thread is not, so the network thread is a bottleneck. In that case set `iproto_threads` to 2 or more. The operating system kernel will determine which connection goes to which thread.

On typical systems, the default value (1) is correct.

Тип: целое число
По умолчанию: 1
Environment variable: TT_IPROTO_THREADS
Динамический: нет

### 9.2.8 Запись в журнал

This section provides information on how to configure options related to logging. You can also use the *log module* to configure logging in your application.

- *log_level*
- *log*
- *log_nonblock*
- *too_long_threshold*
- *log_format*
- *log_modules*

`log_level`
Since version 1.6.2.

Specify the level of detail the *log* has. There are the following levels:

- 0 – `fatal`
- 1 – `syserror`
- 2 – `error`
- 3 – `crit`
- 4 – `warn`
- 5 – `info`
- 6 – `verbose`
- 7 – `debug`

By setting `log_level`, you can enable logging of all events with severities above or equal to the given level. Tarantool prints logs to the standard error stream by default. This can be changed with the *log* configuration parameter.

Type: integer, string
По умолчанию: 5
Environment variable: TT_LOG_LEVEL
Динамический: да

---

**Примечание:** Prior to Tarantool 1.7.5 there were only six levels and `DEBUG` was level 6. Starting with Tarantool 1.7.5, `VERBOSE` is level 6 and `DEBUG` is level 7. `VERBOSE` is a new level for monitoring repetitive events which would cause too much log writing if `INFO` were used instead.

---

`log`

Since version 1.7.4.

By default, Tarantool sends the log to the standard error stream (`stderr`). If `log` is specified, Tarantool can send the log to a:

- file
- pipe
- system logger

Example 1: sending the log to the `tarantool.log` file.

```
box.cfg{log = 'tarantool.log'}
-- или
box.cfg{log = 'file:tarantool.log'}
```

This opens the file `tarantool.log` for output on the server's default directory. If the `log` string has no prefix or has the prefix «file:», then the string is interpreted as a file path.

Example 2: sending the log to a pipe.

```
box.cfg{log = '| cronolog tarantool.log'}
-- or
box.cfg{log = 'pipe: cronolog tarantool.log'}
```

This starts the program cronolog when the server starts, and sends all log messages to the standard input (`stdin`) of `cronolog`. If the `log` string begins with „|“ or has the prefix «pipe:», then the string is interpreted as a Unix pipeline.

Example 3: sending the log to syslog.

```
box.cfg{log = 'syslog:identity=tarantool'}
-- или
box.cfg{log = 'syslog:facility=user'}
-- или
box.cfg{log = 'syslog:identity=tarantool,facility=user'}
-- или
box.cfg{log = 'syslog:server=unix:/dev/log'}
```

If the `log` string begins with «syslog:», then it is interpreted as a message for the syslogd program, which normally is running in the background on any Unix-like platform. The setting can be `syslog:`, `syslog:facility=...`, `syslog:identity=...`, `syslog:server=...`, or a combination.

- The `syslog:identity` setting is an arbitrary string, which is placed at the beginning of all messages. The default value is «tarantool».
- В настоящий момент настройка `syslog:facility` не учитывается, но будет использоваться в дальнейшем. Ее значением должно быть одно из ключевых слов syslog, которые сообщают программе syslogd, куда отправлять сообщение. Возможные значения: auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, security, syslog, user, uucp, local0, local1, local2, local3, local4, local5, local6, local7. По умолчанию: user.
- The `syslog:server` setting is the locator for the syslog server. It can be a Unix socket path beginning with «unix:», or an ipv4 port number. The default socket value is: `dev/log` (on Linux) or `/var/run/syslog` (on macOS). The default port value is: 514, the UDP port.

When logging to a file, Tarantool reopens the log on SIGHUP. When log is a program, its PID is saved in the *log.pid* variable. You need to send it a signal to rotate logs.

Тип: строка

По умолчанию: null

Environment variable: TT_LOG

Динамический: нет

`log_nonblock`

Since version 1.7.4.

If `log_nonblock` equals **true**, Tarantool does not block during logging when the system is not ready for writing, and drops the message instead. If *log_level* is high, and many messages go to the log, setting `log_nonblock` to **true** may improve logging performance at the cost of some log messages getting lost.

This parameter has effect only if *log* is configured to send logs to a pipe or system logger. The default `log_nonblock` value is **nil**, which means that blocking behavior corresponds to the logger type:

- **false** for `stderr` and file loggers.
- **true** for a pipe and system logger.

This is a behavior change: in earlier versions of the Tarantool server, the default value was **true**.

Тип: логический

По умолчанию: nil

Environment variable: TT_LOG_NONBLOCK

Динамический: нет

`too_long_threshold`

Since version 1.6.2.

If processing a request takes longer than the given value (in seconds), warn about it in the log. Has effect only if *log_level* is greater than or equal to 4 (WARNING).

Тип: число с плавающей запятой

По умолчанию: 0.5

Environment variable: TT_TOO_LONG_THRESHOLD

Динамический: да

`log_format`

Since version 1.7.6.

Log entries have two possible formats:

- „plain“ (по умолчанию) или
- „json“ (более детально с JSON-метками).

Here is what a log entry looks like if `box.cfg{log_format='plain'}`:

```
2017-10-16 11:36:01.508 [18081] main/101/interactive I> set 'log_format'␣
↪configuration option to "plain"
```

Here is what a log entry looks like if `box.cfg{log_format='json'}`:

```
{"time": "2017-10-16T11:36:17.996-0600",
"level": "INFO",
"message": "set 'log_format' configuration option to \"json\"",
"pid": 18081,|
"cord_name": "main",
"fiber_id": 101,
"fiber_name": "interactive",
"file": "builtin\/box\/load_cfg.lua",
"line": 317}
```

The `log_format='plain'` entry has a time value, process ID, cord name, *fiber_id*, *fiber_name*, *log level*, and message.

The `log_format='json'` entry has the same fields along with their labels, and in addition has the file name and line number of the Tarantool source.

Тип: строка

По умолчанию: „plain“
Environment variable: TT_LOG_FORMAT
Динамический: да

`log_modules`
Since version *2.11.0*.

Configure the specified log levels (*log_level*) for different modules.

You can specify a logging level for the following module types:

- Modules (files) that use the default logger. Example: *Set log levels for files that use the default logger*.
- Modules that use custom loggers created using the *log.new()* function. Example: *Set log levels for modules that use custom loggers*.
- The `tarantool` module that enables you to configure the logging level for Tarantool core messages. Specifically, it configures the logging level for messages logged from non-Lua code, including C modules. Example: *Set a log level for C modules*.

Type: table
Default: blank
Environment variable: TT_LOG_MODULES
Динамический: да

**Example 1: Set log levels for files that use the default logger**

Suppose you have two identical modules placed by the following paths: `test/logging/module1.lua` and `test/logging/module2.lua`. These modules use the default logger and look as follows:

```
return {
    say_hello = function()
        local log = require('log')
        log.info('Info message from module1')
    end
}
```

To load these modules in your application, you need to add the corresponding `require` directives:

```
module1 = require('test.logging.module1')
module2 = require('test.logging.module2')
```

To configure logging levels, you need to provide module names corresponding to paths to these modules. In the example below, the `box_cfg` variable contains logging settings that can be passed to the `box.cfg()` function:

```
box_cfg = { log_modules = {
    ['test.logging.module1'] = 'verbose',
    ['test.logging.module2'] = 'error' }
}
```

Given that `module1` has the `verbose` logging level and `module2` has the `error` level, calling `module1.say_hello()` shows a message but `module2.say_hello()` is swallowed:

```
-- Prints 'info' messages --
module1.say_hello()
--[[
[92617] main/103/interactive/test.logging.module1 I> Info message from module1
---
...
--]]

-- Swallows 'info' messages --
module2.say_hello()
--[[
---
...
--]]
```

**Example 2: Set log levels for modules that use custom loggers**

In the example below, the `box_cfg` variable contains logging settings that can be passed to the `box.cfg()` function. This example shows how to set the `verbose` level for `module1` and the `error` level for `module2`:

```
box_cfg = { log_level = 'warn',
            log_modules = {
                module1 = 'verbose',
                module2 = 'error' }
}
```

To create custom loggers, call the *log.new()* function:

```
-- Creates new loggers --
module1_log = require('log').new('module1')
module2_log = require('log').new('module2')
```

Given that `module1` has the `verbose` logging level and `module2` has the `error` level, calling `module1_log.info()` shows a message but `module2_log.info()` is swallowed:

```
-- Prints 'info' messages --
module1_log.info('Info message from module1')
--[[
[16300] main/103/interactive/module1 I> Info message from module1
---
...
--]]

-- Swallows 'debug' messages --
module1_log.debug('Debug message from module1')
--[[
---
...
--]]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Swallows 'info' messages --
module2_log.info('Info message from module2')
--[[
---
...
--]]
```

**Example 3: Set a log level for C modules**

In the example below, the `box_cfg` variable contains logging settings that can be passed to the `box.cfg()` function. This example shows how to set the `info` level for the `tarantool` module:

```
box_cfg = { log_level = 'warn',
            log_modules = { tarantool = 'info' } }
```

The specified level affects messages logged from C modules:

```
ffi = require('ffi')

-- Prints 'info' messages --
ffi.C._say(ffi.C.S_INFO, nil, 0, nil, 'Info message from C module')
--[[
[6024] main/103/interactive I> Info message from C module
---
...
--]]

-- Swallows 'debug' messages --
ffi.C._say(ffi.C.S_DEBUG, nil, 0, nil, 'Debug message from C module')
--[[
---
...
--]]
```

The example above uses the LuaJIT ffi library to call C functions provided by the `say` module.

### Пример записи в журнал

This example illustrates how «rotation» works, that is, what happens when the server instance is writing to a log and signals are used when archiving it.

1. Start with two terminal shells: Terminal #1 and Terminal #2.
2. In Terminal #1, start an interactive Tarantool session. Then, use the `log` property to send logs to `Log_file` and call `log.info` to put a message in the log file.

```
box.cfg{log='Log_file'}
log = require('log')
log.info('Log Line #1')
```

3. In Terminal #2, use the `mv` command to rename the log file to `Log_file.bak`.

```
mv Log_file Log_file.bak
```

As a result, the next log message will go to `Log_file.bak`.

4. Go back to Terminal #1 and put a message «Log Line #2» in the log file.

```
log.info('Log Line #2')
```

5. In Terminal #2, use `ps` to find the process ID of the Tarantool instance.

```
ps -A | grep tarantool
```

6. In Terminal #2, execute `kill -HUP` to send a SIGHUP signal to the Tarantool instance. Tarantool will open `Log_file` again, and the next log message will go to `Log_file`.

   `kill -HUP` *`process_id`*

   The same effect could be accomplished by calling *log.rotate*.

7. In Terminal #1, put a message «Log Line #3» in the log file.

```
log.info('Log Line #3')
```

8. In Terminal #2, use `less` to examine files. `Log_file.bak` will have the following lines . . .

```
2015-11-30 15:13:06.373 [27469] main/101/interactive I> Log Line #1`
2015-11-30 15:14:25.973 [27469] main/101/interactive I> Log Line #2`
```

   . . . and `Log_file` will look like this:

```
log file has been reopened
2015-11-30 15:15:32.629 [27469] main/101/interactive I> Log Line #3
```

### 9.2.9 Audit log

**Enterprise Edition**

Audit log features are available in the Enterprise Edition only.

The `audit_*` parameters define configuration related to *audit logging*.

- *audit_extract_key*
- *audit_filter*
- *audit_format*
- *audit_log*
- *audit_nonblock*
- *audit_spaces*

`audit_extract_key`

**Since:** *3.0.0*.

If set to `true`, the audit subsystem extracts and prints only the primary key instead of full tuples in DML events (`space_insert`, `space_replace`, `space_delete`). Otherwise, full tuples are logged. The option may be useful in case tuples are big.

Тип: логический

По умолчанию: false (ложь)

Environment variable: TT_AUDIT_EXTRACT_KEY

`audit_filter`

Enable logging for a specified subset of audit events. This option accepts the following values:

- Event names (for example, `password_change`). For details, see *Audit log events*.
- Event groups (for example, `audit`). For details, see *Event groups*.

The option contains either one value from `Possible values` section (see below) or a combination of them.

To enable *custom audit log events*, specify the `custom` value in this option.

The default value is `compatibility`, which enables logging of all events available before 2.10.0.

**Example**

```
box.cfg{
    audit_log = 'audit.log',
    audit_filter = 'audit,auth,priv,password_change,access_denied'
   }
```

Type: array

Possible values: „all“, „audit“, „auth“, „priv“, „ddl“, „dml“, „data_operations“, „compatibility“, „audit_enable“, „auth_ok“, „auth_fail“, „disconnect“, „user_create“, „user_drop“, „role_create“, „role_drop“, „user_disable“, „user_enable“, „user_grant_rights“, „role_grant_rights“, „role_revoke_rights“, „password_change“, „access_denied“, „eval“, „call“, „space_select“, „space_create“, „space_alter“, „space_drop“, „space_insert“, „space_replace“, „space_delete“, „custom“

Default: „compatibility“

Environment variable: TT_AUDIT_FILTER

`audit_format`

Specify the format that is used for the audit log events – plain text, CSV or JSON format.

Plain text is used by default. This human-readable format can be efficiently compressed.

```
box.cfg{audit_log = 'audit.log', audit_format = 'plain'}
```

**Example**

```
remote: session_type:background module:common.admin.auth user: type:custom_tdg_
↪audit tag:tdg_severity_INFO description:[5e35b406-4274-4903-857b-c80115275940]␣
↪subj: "anonymous", msg: "Access granted to anonymous user"
```

The JSON format is more convenient to receive log events, analyze them and integrate them with other systems if needed.

```
box.cfg{audit_log = 'audit.log', audit_format = 'json'}
```

**Example**

```
{"time": "2022-11-17T21:55:49.880+0300", "remote": "", "session_type": "background",
↪ "module": "common.admin.auth", "user": "", "type": "custom_tdg_audit", "tag":
↪"tdg_severity_INFO", "description": "[c26cd11a-3342-4ce6-8f0b-a4b222268b9d] subj:␣
↪\"anonymous\", msg: \"Access granted to anonymous user\""}
```

Using the CSV format allows you to view audit log events in tabular form.

```
box.cfg{audit_log = 'audit.log', audit_format = 'csv'}
```

**Example**

```
2022-11-17T21:58:03.131+0300,,background,common.admin.auth,,,custom_tdg_audit,tdg_
↪severity_INFO,"[b3dfe2a3-ec29-4e61-b747-eb2332c83b2e] subj: ""anonymous"", msg: "
↪"Access granted to anonymous user"""
```

Тип: строка
Possible values: „json“, „csv“, „plain“
Default: „json“
Environment variable: TT_AUDIT_FORMAT

`audit_log`

Enable audit logging and define the log location.

This option accepts a string value that allows you to define the log location. The following locations are supported:

- File: to write audit logs to a file, specify a path to a file (with an optional `file` prefix)
- Pipeline: to start a program and write audit logs to it, specify a program name (with `|` or `pipe` prefix)
- System log: to write audit logs to a system log, specify a message for `syslogd` (with `syslog` prefix)

See the examples below.

By default, audit logging is disabled.

**Example: Writing to a file**

```
box.cfg{audit_log = 'audit_tarantool.log'}
-- or
box.cfg{audit_log = 'file:audit_tarantool.log'}
```

This opens the `audit_tarantool.log` file for output in the server's default directory. If the `audit_log` string has no prefix or the prefix `file:`, the string is interpreted as a file path.

If you log to a file, Tarantool will reopen the audit log at SIGHUP.

**Example: Sending to a pipeline**

```
box.cfg{audit_log = '| cronolog audit_tarantool.log'}
-- or
box.cfg{audit_log = 'pipe: cronolog audit_tarantool.log'}'
```

This starts the cronolog program when the server starts and sends all `audit_log` messages to cronolog's standard input (`stdin`). If the `audit_log` string starts with „|“ or contains the prefix `pipe:`, the string is interpreted as a Unix pipeline.

If log is a program, check out its pid and send it a signal to rotate logs.

**Example: Writing to a system log**

**Предупреждение:** Below is an example of writing audit logs to a directory shared with the system logs. Tarantool allows this option, but it is **not recommended** to do this to avoid difficulties when working with audit logs. System and audit logs should be written separately. To do this, create separate paths and specify them.

This sample configuration sends the audit log to syslog:

```
box.cfg{audit_log = 'syslog:identity=tarantool'}
-- or
box.cfg{audit_log = 'syslog:facility=user'}
-- or
box.cfg{audit_log = 'syslog:identity=tarantool,facility=user'}
-- or
box.cfg{audit_log = 'syslog:server=unix:/dev/log'}
```

If the `audit_log` string starts with «syslog:», it is interpreted as a message for the syslogd program, which normally runs in the background of any Unix-like platform. The setting can be „syslog:“, „syslog:facility=...“, „syslog:identity=...“, „syslog:server=...“ or a combination.

The `syslog:identity` setting is an arbitrary string that is placed at the beginning of all messages. The default value is `tarantool`.

The `syslog:facility` setting is currently ignored, but will be used in the future. The value must be one of the syslog keywords that tell `syslogd` where to send the message. The possible values are `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `security`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, `local7`. The default value is `local7`.

The `syslog:server` setting is the locator for the syslog server. It can be a Unix socket path starting with «unix:» or an ipv4 port number. The default socket value is `/dev/log` (on Linux) or `/var/run/syslog` (on Mac OS). The default port value is 514, which is the UDP port.

An example of a Tarantool audit log entry in the syslog:

```
09:32:52 tarantool_audit: {"time": "2024-02-08T09:32:52.190+0300", "uuid":
↪"94454e46-9a0e-493a-bb9f-d59e44a43581", "severity": "INFO", "remote": "unix/
↪:(socket)", "session_type": "console", "module": "tarantool", "user": "admin",
↪"type": "space_create", "tag": "", "description": "Create space bands"}
```

Тип: строка

Possible values: see the string format above

Default: „nill“
Environment variable: TT_AUDIT_LOG

`audit_nonblock`
Specify the logging behavior if the system is not ready to write. If set to `true`, Tarantool does not block during logging if the system is non-writable and writes a message instead. Using this value may improve logging performance at the cost of losing some log messages.

---

**Примечание:** The option only has an effect if the *audit_log* is set to `syslog` or `pipe`.

Setting `audit_nonblock` to `true` is not allowed if the output is to a file. In this case, set `audit_nonblock` to `false`.

---

Тип: логический
По умолчанию: true
Environment variable: TT_AUDIT_NONBLOCK

`audit_spaces`
**Since:** *3.0.0*.

The array of space names for which data operation events (`space_select`, `space_insert`, `space_replace`, `space_delete`) should be logged. The array accepts string values. If set to *box.NULL*, the data operation events are logged for all spaces.

**Example**

In the example, only the events of `bands` and `singers` spaces are logged:

```
box.cfg{
    audit_spaces = 'bands,singers'
   }
```

Type: array
Default: box.NULL
Environment variable: TT_AUDIT_SPACES

### 9.2.10 Authentication

---

**Enterprise Edition**

Authentication features are supported by the Enterprise Edition only.

---

- *auth_delay*
- *auth_retries*

- *auth_type*
- *disable_guest*
- *password_min_length*
- *password_enforce_uppercase*
- *password_enforce_lowercase*
- *password_enforce_digits*
- *password_enforce_specialchars*
- *password_lifetime_days*
- *password_history_length*

`auth_delay`

Since *2.11.0*.

Specify a period of time (in seconds) that a specific user should wait for the next attempt after failed authentication.

With the configuration below, Tarantool refuses the authentication attempt if the previous attempt was less than 5 seconds ago.

```
box.cfg{ auth_delay = 5 }
```

Тип: число

Default: 0

Environment variable: TT_AUTH_DELAY

Динамический: да

`auth_retries`

Since *3.0.0*.

Specify the maximum number of authentication retries allowed before `auth_delay` is enforced. The default value is 0, which means `auth_delay` is enforced after the first failed authentication attempt.

The retry counter is reset after `auth_delay` seconds since the first failed attempt. For example, if a client tries to authenticate fewer than `auth_retries` times within `auth_delay` seconds, no authentication delay is enforced. The retry counter is also reset after any successful authentication attempt.

Тип: число

Default: 0

Environment variable: TT_AUTH_RETRIES

Динамический: да

`auth_type`

Since *2.11.0*.

Specify an authentication protocol:

- „chap-sha1“: use the CHAP protocol to authenticate users with `SHA-1` hashing applied to *passwords*.
- „pap-sha256“: use PAP authentication with the `SHA256` hashing algorithm.

For new users, the *box.schema.user.create* method will generate authentication data using `PAP-SHA256`. For existing users, you need to reset a password using *box.schema.user.passwd* to use the new authentication protocol.

Тип: строка
Default value: „chap-sha1“
Environment variable: TT_AUTH_TYPE
Динамический: да

`disable_guest`

Since *2.11.0*.

If **true**, disables access over remote connections from unauthenticated or *guest access* users. This option affects both *net.box* and *replication* connections.

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_DISABLE_GUEST
Динамический: да

`password_min_length`

Since *2.11.0*.

Specify the minimum number of characters for a password.

The following example shows how to set the minimum password length to 10.

```
box.cfg{ password_min_length = 10 }
```

Тип: целое число
Default: 0
Environment variable: TT_PASSWORD_MIN_LENGTH
Динамический: да

`password_enforce_uppercase`

Since *2.11.0*.

If **true**, a password should contain uppercase letters (A-Z).

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_PASSWORD_ENFORCE_UPPERCASE
Динамический: да

`password_enforce_lowercase`
Since *2.11.0*.

If **true**, a password should contain lowercase letters (a-z).

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_PASSWORD_ENFORCE_LOWERCASE
Динамический: да

`password_enforce_digits`
Since *2.11.0*.

If **true**, a password should contain digits (0-9).

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_PASSWORD_ENFORCE_DIGITS
Динамический: да

`password_enforce_specialchars`
Since *2.11.0*.

If **true**, a password should contain at least one special character (such as `&|?!@$`).

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_PASSWORD_ENFORCE_SPECIALCHARS
Динамический: да

`password_lifetime_days`
Since *2.11.0*.

Specify the maximum period of time (in days) a user can use the same password. When this period ends, a user gets the «Password expired» error on a login attempt. To restore access for such users, use *box.schema.user.passwd*.

**Примечание:** The default 0 value means that a password never expires.

The example below shows how to set a maximum password age to 365 days.

```
box.cfg{ password_lifetime_days = 365 }
```

Тип: целое число
Default: 0
Environment variable: TT_PASSWORD_LIFETIME_DAYS
Динамический: да

`password_history_length`
Since *2.11.0*.

Specify the number of unique new user passwords before an old password can be reused.

In the example below, a new password should differ from the last three passwords.

```
box.cfg{ password_history_length = 3 }
```

Тип: целое число
Default: 0
Environment variable: TT_PASSWORD_HISTORY_LENGTH
Динамический: да

**Примечание:** Tarantool uses the `auth_history` field in the *box.space._user* system space to store user passwords.

### 9.2.11 Flight recorder

**Enterprise Edition**

The flight recorder is available in the Enterprise Edition only.

- *flightrec_enabled*
- *flightrec_logs_size*
- *flightrec_logs_max_msg_size*
- *flightrec_logs_log_level*
- *flightrec_metrics_period*

- *flightrec_metrics_interval*
- *flightrec_requests_size*
- *flightrec_requests_max_req_size*
- *flightrec_requests_max_res_size*

`flightrec_enabled`
Since *2.11.0*.

Enable the *flight recorder*.

Тип: логический
По умолчанию: false (ложь)
Environment variable: TT_FLIGHTREC_ENABLED
Динамический: да

`flightrec_logs_size`
Since *2.11.0*.

Specify the size (in bytes) of the log storage. You can set this option to `0` to disable the log storage.

Тип: целое число
Default: 10485760
Environment variable: TT_FLIGHTREC_LOGS_SIZE
Динамический: да

`flightrec_logs_max_msg_size`
Since *2.11.0*.

Specify the maximum size (in bytes) of the log message. The log message is truncated if its size exceeds this limit.

Тип: целое число
Default: 4096
Maximum: 16384
Environment variable: TT_FLIGHTREC_LOGS_MAX_MSG_SIZE
Динамический: да

`flightrec_logs_log_level`
Since *2.11.0*.

Specify the level of detail the log has. You can learn more about log levels from the *log_level* option description. Note that the `flightrec_logs_log_level` value might differ from `log_level`.

Тип: целое число

Default: 6

Environment variable: TT_FLIGHTREC_LOGS_LOG_LEVEL

Динамический: да

`flightrec_metrics_period`

Since *2.11.0*.

Specify the time period (in seconds) that defines how long metrics are stored from the moment of dump. So, this value defines how much historical metrics data is collected up to the moment of crash. The frequency of metric dumps is defined by *flightrec_metrics_interval*.

Тип: целое число

Default: 180

Environment variable: TT_FLIGHTREC_METRICS_PERIOD

Динамический: да

`flightrec_metrics_interval`

Since *2.11.0*.

Specify the time interval (in seconds) that defines the frequency of dumping metrics. This value shouldn't exceed *flightrec_metrics_period*.

---

**Примечание:** Given that the average size of a metrics entry is 2 kB, you can estimate the size of the metrics storage as follows:

```
(flightrec_metrics_period / flightrec_metrics_interval) * 2 kB
```

---

Тип: число

По умолчанию: 1.0

Minimum: 0.001

Environment variable: TT_FLIGHTREC_METRICS_INTERVAL

Динамический: да

`flightrec_requests_size`

Since *2.11.0*.

Specify the size (in bytes) of storage for the request and response data. You can set this parameter to `0` to disable a storage of requests and responses.

Тип: целое число

Default: 10485760
Environment variable: TT_FLIGHTREC_REQUESTS_SIZE
Динамический: да

`flightrec_requests_max_req_size`
Since *2.11.0*.

Specify the maximum size (in bytes) of a request entry. A request entry is truncated if this size is exceeded.

Тип: целое число
Default: 16384
Environment variable: TT_FLIGHTREC_REQUESTS_MAX_REQ_SIZE
Динамический: да

`flightrec_requests_max_res_size`
Since *2.11.0*.

Specify the maximum size (in bytes) of a response entry. A response entry is truncated if this size is exceeded.

Тип: целое число
Default: 16384
Environment variable: TT_FLIGHTREC_REQUESTS_MAX_RES_SIZE
Динамический: да

### 9.2.12 Обратная связь

- *feedback_enabled*
- *feedback_host*
- *feedback_interval*

By default, a Tarantool daemon sends a small packet once per hour, to `https://feedback.tarantool.io`. The packet contains three values from *box.info*: `box.info.version`, `box.info.uuid`, and `box.info.cluster_uuid`. By changing the feedback configuration parameters, users can adjust or turn off this feature.

`feedback_enabled`
Since version 1.10.1.

Whether to send feedback.

Если задано значение `true`, обратная связь будет отправлена, как описано выше. Если задано значение `false`, обратная связь не отправляется.

Тип: логический

По умолчанию: true
Environment variable: TT_FEEDBACK_ENABLED
Динамический: да

`feedback_host`
Since version 1.10.1.

The address to which the packet is sent. Usually the recipient is Tarantool, but it can be any URL.

Тип: строка
Default: `https://feedback.tarantool.io`
Environment variable: TT_FEEDBACK_HOST
Динамический: да

`feedback_interval`
Since version 1.10.1.

The number of seconds between sendings, usually 3600 (1 hour).

Тип: число с плавающей запятой
По умолчанию: 3600
Environment variable: TT_FEEDBACK_INTERVAL
Динамический: да

### 9.2.13 Устаревшие параметры

Данные параметры объявлены устаревшими с версии Tarantool 1.7.4:

- *coredump*
- *logger*
- *logger_nonblock*
- *panic_on_snap_error*,
- *panic_on_wal_error*
- *replication_source*
- *slab_alloc_arena*
- *slab_alloc_maximal*
- *slab_alloc_minimal*
- *snap_dir*
- *snapshot_count*
- *snapshot_period*
- *rows_per_wal*,

- *election_fencing_enabled*

`coredump`
**Устаревший**, не использовать.

Тип: логический
По умолчанию: false (ложь)
Динамический: нет

`logger`
**Устаревший**, заменен параметром *log*. Параметр был лишь переименован, его тип, значения и семантика остались прежними.

`logger_nonblock`
**Устаревший**, заменен параметром *log_nonblock*. Параметр был лишь переименован, его тип, значения и семантика остались прежними.

`panic_on_snap_error`
**Устаревший**, заменен параметром *force_recovery*.

Если при чтении файла снимка произошла ошибка (при запуске экземпляра сервера), прервать выполнение.

Тип: логический
По умолчанию: true
Динамический: нет

`panic_on_wal_error`
**Устаревший**, заменен параметром *force_recovery*.

Тип: логический
По умолчанию: true
Динамический: да

`replication_source`
**Устаревший**, заменен параметром *replication*. Параметр был лишь переименован, его тип, значения и семантика остались прежними.

`slab_alloc_arena`
**Устаревший**, заменен параметром *memtx_memory*.

Количество памяти, которое Tarantool выделяет для фактического хранения кортежей, **в гигабайтах**. При достижении предельного значения запросы вставки INSERT или обновления UPDATE выполняться не будут, выдавая ошибку `ER_MEMORY_ISSUE`. Сервер не выходит за установленный предел памяти `memtx_memory` при распределении кортежей, но есть дополнительная

память, которая используется для хранения индексов и информации о подключении. В зависимости от рабочей конфигурации и загрузки, Tarantool может потреблять на 20% больше установленного предела.

Тип: число с плавающей запятой

По умолчанию: 1.0

Динамический: нет

`slab_alloc_maximal`
**Устаревший**, заменен параметром *memtx_max_tuple_size*. Параметр был лишь переименован, его тип, значения и семантика остались прежними.

`slab_alloc_minimal`
**Устаревший**, заменен параметром *memtx_min_tuple_size*. Параметр был лишь переименован, его тип, значения и семантика остались прежними.

`snap_dir`
**Устаревший**, заменен параметром *memtx_dir*. Параметр был лишь переименован, его тип, значения и семантика остались прежними.

`snapshot_period`
**Устаревший**, заменен параметром *checkpoint_interval*. Параметр был лишь переименован, его тип, значения и семантика остались прежними.

`snapshot_count`
**Устаревший**, заменен параметром *checkpoint_count*. Параметр был лишь переименован, его тип, значения и семантика остались прежними.

`rows_per_wal`
**Deprecated** in favor of *wal_max_size*. The parameter does not allow to properly limit size of WAL logs.

`election_fencing_enabled`
**Deprecated** in Tarantool v2.11 in favor of *election_fencing_mode*.

The parameter does not allow using the `strict` fencing mode. Setting to `true` is equivalent to setting the `soft` *election_fencing_mode*. Setting to `false` is equivalent to setting the `off` *election_fencing_mode*.

Тип: логический

По умолчанию: true

Environment variable: TT_ELECTION_FENCING_ENABLED

Динамический: да

## 9.3 Tooling

This chapter describes the tools that enable developers and administrators to work with Tarantool.

### 9.3.1 tt CLI utility

`tt` is a utility that provides a unified command-line interface for managing Tarantool-based applications. It covers a wide range of tasks – from installing a specific Tarantool version to managing remote instances and developing applications.

`tt` is developed in its own GitHub repository. Here you can find its source code, changelog, and releases information. For a complete list of releases, see the Releases section on GitHub.

There is also the Enterprise version of `tt` available in a Tarantool Enterprise Edition's *release package*. The Enterprise version provides additional features, for example, *importing* and *exporting* data.

This section provides instructions on `tt` installation and configuration, concept explanation, and the `tt` command reference.

#### Installation

To install the `tt` command-line utility, use a package manager – Yum or APT on Linux, or Homebrew on macOS. If you need a specific build, you can build `tt` from sources.

---

**Примечание:** A Tarantool Enterprise Edition's *release package* includes the `tt` utility extended with additional features like *importing* and *exporting* data.

---

#### Using Linux package managers

On Linux systems, you can install `tt` with `yum` or `apt` package managers from the `tarantool/modules` repository. Learn how to add this repository.

The installation command looks like this:

- On Ubuntu:

```
$ sudo apt-get install tt
```

- On CentOS:

```
$ sudo yum install tt
```

### Using Homebrew on macOS

On macOS, use Homebrew to install `tt`:

```
$ brew install tt
```

### Building from sources

To build `tt` from sources:

1. Install third-party software required for building `tt`:
   - git, the version control system.
   - Go language, version 1.18 or later.
   - mage build tool.
2. Clone the tarantool/tt repository:

   ```
   git clone https://github.com/tarantool/tt --recursive
   ```

3. Go to the `tt` directory:

   ```
   cd tt
   ```

4. (Optional) Checkout a release tag to build a specific version:

   ```
   git checkout tags/v1.0.0
   ```

5. Build `tt` using mage:

   ```
   mage build
   ```

`tt` will appear in the current directory.

### Enabling shell completion

To enable the completion for `tt` commands, run the following command specifying the shell (`bash` or `zsh`):

```
. <(tt completion bash)
```

## Configuration

### Configuration file

The key artifact that defines the `tt` environment and various aspects of its execution is its configuration file.

By default, the configuration file is called `tt.yaml`. The location where `tt` searches for it depends on the *launch mode*. You can also pass the configuration file explicitly in the `--cfg` *global option*.

The `tt` configuration file is a YAML file with the following structure:

```
env:
  instances_enabled: path/to/available/applications
  bin_dir: path/to/bin_dir
  inc_dir: path/to/inc_dir
  restart_on_failure: bool
  tarantoolctl_layout: bool
modules:
  directory: path/to/modules/dir
app:
  run_dir: path/to/run_dir
  log_dir: path/to/log_dir
  wal_dir: path/to/wal_dir
  vinyl_dir: path/to/vinyl_dir
  memtx_dir: path/to/memtx_dir
repo:
  rocks: path/to/rocks
  distfiles: path/to/install
ee:
  credential_path: path/to/file
templates:
  - path: path/to/app/templates1
  - path: path/to/app/templates2
```

**Примечание:** The `tt` configuration format and application layout have been changed in version 2.0. Learn how to upgrade from earlier versions in *Migrating from tt 1.* to 2.0 or later*.

#### env section

**Примечание:** The paths specified in `env.*` parameters are relative to the current `tt` environment's root.

- `instances_enabled` – the directory where *instances* are stored. Default: `instances.enabled`.
- `bin_dir` – the directory where binary files are stored. Default: `bin`.
- `inc_dir` – the base directory for storing header files. They will be placed in the `include` subdirectory inside the specified directory. Default: `include`.

  **Примечание:** The header files directory path can also be passed using the `TT_CLI_TARANTOOL_PREFIX` environment variable. If it is set, `tt rocks` and `tt build` commands use the `include/tarantool` directory inside `TT_CLI_TARANTOOL_PREFIX` as the header files directory.

- `restart_on_failure` – restart the instance on failure: `true` or `false`. Default: `false`.
- `tarantoolctl_layout` – use a layout compatible with the deprecated `tarantoolctl` utility for artifact files: control sockets, `.pid` files, log files. Default: `false`.

#### modules section

- `directory` – the directory where *external modules* are stored.

#### app section

---

**Примечание:** The paths specified in `app.*_dir` parameters are relative to the application location inside the `instances.enabled` directory specified in the `env` configuration section. For example, the default location of the `myapp` application's logs is `instances.enabled/myapp/var/log`. Inside this location, `tt` creates separate directories for each application instance that runs in the current environment.

---

- `run_dir`– the directory for instance runtime artifacts, such as console sockets or PID files. Default: `var/run`.
- `log_dir` – the directory where log files are stored. Default: `var/log`.
- `wal_dir` – the directory where write-ahead log (`.xlog`) files are stored. Default: `var/lib`.
- `memtx_dir` – the directory where memtx stores snapshot (`.snap`) files. Default: `var/lib`.
- `vinyl_dir` – the directory where vinyl files or subdirectories are stored. Default: `var/lib`.

#### repo section

- `rocks` – the directory where rocks files are stored.

  ---

  **Примечание:** The rocks directory path can be passed in the `TT_CLI_REPO_ROCKS` environment variable instead. The variable is also used if the directory specified in `repo.rocks` does not include a repository manifest.

  ---

- `distfiles` – the directory where installation files are stored.

#### ee section

- `credential_path` – a path to the file with credentials used for downloading Tarantool Enterprise Edition (Tarantool customer zone credentials). The file should contain a username and a password, each on a separate line. Find an example in the *tt install* command reference.

  ---

  **Примечание:** The customer zone credentials can also be passed in the `TT_CLI_EE_USERNAME` and `TT_CLI_EE_PASSWORD` environment variables.

  ---

#### templates section

- `path` – a path to application templates used for creating applications with *tt create*. May be specified more than once.

### Launch modes

`tt` launch mode defines its working directory and the way it searches for the configuration file. There are three launch modes:

- default
- system
- local

#### Default launch

**Global option**: none

**Configuration file**: searched from the current directory to the root. Taken from `/etc/tarantool` if the file is not found.

**Working directory**: The directory where the configuration file is found.

#### System launch

**Global option**: `--system` or `-S`

**Configuration file**: Taken from `/etc/tarantool`.

**Working directory**: Current directory.

#### Local launch

**Global option**: `--local=DIRECTORY` or `-L=DIRECTORY`

**Configuration file**: Searched from the specified directory to the root. Taken from `/etc/tarantool` if the file is not found.

**Working directory**: The specified directory. If `tarantool` or `tt` executable files are found in the working directory, they will be used.

### Migrating from tt 1.* to 2.0 or later

The `tt` configuration and application layout were changed in version 2.0. If you are using `tt` 1.*, complete the following steps to migrate to `tt` 2.0 or later:

1. **Update the tt configuration file**. In tt 2.0, the following changes were made to the configuration file:
   - The root section `tt` was removed. Its child sections – `app`, `repo`, `modules`, and other – have been moved to the top level.

- Environment configuration parameters were moved from the `app` section to the new section `env`. These parameters are `instances.enabled`, `bin_dir`, `inc_dir`, and `restart_on_failure`.
- The paths in the `app` section are now relative to the app directory in `instances.enabled` instead of the environment root.

You can use *tt init* to generate a configuration file with the new structure and default parameter values.

2. **Move application artifacts**. With `tt` 1.*, application artifacts (logs, snapshots, pid, and other files) were created in the `var` directory inside the *environment root*. Starting from `tt` 2.0, these artifacts are created in the `var` directory inside the *application directory*, which is `instances.enabled/<app-name>`. This is how an application directory looks:

```
instances.enabled/app/
├── init.lua
├── instances.yml
└── var
    ├── lib
    │   ├── instance1
    │   └── instance2
    ├── log
    │   ├── instance1
    │   └── instance2
    └── run
        ├── instance1
        └── instance2
```

   To continue using existing application artifacts after migration from `tt` 1.*:

   1. Create the `var` directory inside the application directory.
   2. Create the `lib`, `log`, and `run` directories inside `var`.
   3. Move directories with instance artifacts from the old `var` directory to the new `var` directories in applications" directories.

3. **Move the files accessed from the application code**. The working directory of instance processes was changed from the `tt` working directory to the application directory inside `instances.enabled`. If the application accesses files using relative paths, move the files accordingly or adjust the application code.

### Global options

**Важно:** Global options of `tt` must be passed before its commands and other options. For example:

```
$ tt --cfg tt-conf.yaml start app
```

`tt` has the following global options:

`-c=FILE, --cfg=FILE,`
: Path to the *configuration file*.

`-h, --help`
: Display help.

`-I, --internal`
: Force the use of an internal module even if there is an *external module* with the same name.

`-L=DIRECTORY, --local=DIRECTORY`
: Use the `tt` environment from the specified directory. Learn more about the *local launch mode*.

`-S, --system`
: Use the `tt` environment installed in the system. Learn more about the *system launch mode*.

`-V, --verbose`
: Display detailed processing information (verbose mode).

### Commands

Below is a list of `tt` commands. Run `tt COMMAND help` to see the detailed help for the given command.

| | |
|---|---|
| *binaries* | Show a list of installed binaries and their versions |
| *build* | Build an application locally |
| *cartridge* | Manage a Cartridge application |
| *cat* | Print the contents of `.snap` or `.xlog` files into stdout |
| *cfg* | Manage a `tt` environment configuration |
| *check* | Check an application file for syntax errors |
| *clean* | Clean instance files |
| *cluster* | Manage a cluster's configuration |
| *completion* | Generate completion for a specified shell |
| *connect* | Connect to a Tarantool instance |
| *coredump* | Manipulate Tarantool core dumps |
| *create* | Create an application from a template |
| *crud* | Interact with the CRUD module (Enterprise only) |
| *export* | Export data to a file (Enterprise only) |
| *help* | Display help for `tt` or a specific command |
| *import* | Import data from a file (Enterprise only) |
| *init* | Create a new `tt` environment in the current directory |
| *install* | Install Tarantool or `tt` |
| *instances* | List enabled applications |
| *logrotate* | Rotate instance logs |
| *pack* | Package an application |
| *play* | Play the contents of `.snap` or `.xlog` files to another Tarantool instance |
| *restart* | Restart a Tarantool instance |
| *rocks* | Use the LuaRocks package manager |
| *run* | Run Lua code in a Tarantool instance |
| *search* | Search available Tarantool and `tt` versions |
| *start* | Start a Tarantool instance |
| *status* | Get the current status of a Tarantool instance |
| *stop* | Stop a Tarantool instance |
| *tdg2* | Interact with Tarantool Data Grid 2 clusters |
| *uninstall* | Uninstall Tarantool or `tt` |
| *version* | Show the `tt` version information |

### Showing a list of installed binaries

```
$ tt binaries
```

`tt binaries` shows a list of installed binaries and their versions.

### Examples

Show a list of installed Tarantool versions:

```
$ tt binaries
List of installed binaries:
   • tarantool:
        2.11.1 [active]
        2.10.8
```

### Building an application

```
$ tt build [PATH] [--spec SPEC_FILE_PATH]
```

`tt build` builds a Tarantool application locally.

### Options

`--spec SPEC_FILE_PATH`
: Path to a `.rockspec` file to use for the current build

### Details

The `PATH` argument should contain the path to the application directory (that is, to the build source). The default path is `.` (current directory).

The application directory must contain a `.rockspec` file to use for the build. If there is more than one `.rockspec` file in the application directory, specify the one to use in the `--spec` argument.

`tt build` builds an application with the `tt rocks make` command. It downloads the application dependencies into the `.rocks` directory, making the application ready to run locally.

### Pre-build and post-build scripts

In addition to building the application with LuaRocks, `tt build` can execute *pre-build* and *post-build* scripts. These scripts should contain steps to execute right before and after building the application. These files must be named `tt.pre-build` and `tt.post-build` correspondingly and located in the application directory.

---

**Примечание:** For compatibility with Cartridge applications, the pre-build and post-build scripts can also have names `cartridge.pre-build` and `cartridge.post-build`.

---

`tt.pre-build` is helpful when your application depends on closed-source rocks, or if the build should contain rocks from a project added as a submodule. You can **install** these dependencies using the pre-build script **before** building. Example:

```
#!/bin/sh

# The main purpose of this script is to build non-standard rocks modules.
# The script will run before `tt rocks make` during application build.

tt rocks make --chdir ./third_party/proj
```

`tt.post-build` is a script that runs after `tt rocks make`. The main purpose of this script is to remove build artifacts from the final package. Example:

```
#!/bin/sh

# The main purpose of this script is to remove build artifacts from the resulting␣
↪package.
# The script will run after `tt rocks make` during application build.

rm -rf third_party
rm -rf node_modules
rm -rf doc
```

### Examples

- Build the application `app1` from its directory:

  ```
  $ tt build
  ```

- Build the application `app1` from the `simple_app` directory inside the current directory:

  ```
  $ tt build simple_app
  ```

- Build the application `app1` from its directory explicitly specifying the rockspec file to use:

  ```
  $ tt build --spec app1-scm-1.rockspec
  ```

### Managing a Cartridge application

---

**Важно:** The Tarantool Cartridge framework is deprecated and is not compatible with Tarantool 3.0 and later. This command is added for backward compatibility with earlier versions.

---

```
$ tt cartridge COMMAND {[OPTION ...]|SUBCOMMAND}
```

`tt cartridge` manages a Cartridge application. `COMMAND` is one of the following:

- *admin*
- *bench*
- *failover*

- *repair*
- *replicasets*

### admin

```
$ tt cartridge admin ADMIN_FUNC_NAME [ADMIN_OPTION ...]
```

`tt cartridge admin` calls admin functions provided by the application.

### Options

`--name STRING`
: (Required) An application name.

`-l, --list`
: List the available admin functions.

`--instance STRING`
: A name of the instance to connect to.

`--conn STRING`
: An address to connect to.

`--run-dir STRING`
: A directory where PID and socket files are stored. Defaults to `/var/run/tarantool`.

### Examples

Get a list of the available admin functions:

```
$ tt cartridge admin --name APPNAME --list

   • Available admin functions:

probe  Probe instance
```

Get help for a specific function:

```
$ tt cartridge admin --name APPNAME probe --help

   • Admin function "probe" usage:

Probe instance

Args:
  --uri string  Instance URI
```

Call a function with an argument:

```
$ tt cartridge admin --name APPNAME probe --uri localhost:3301

   • Probe "localhost:3301": OK
```

### bench

```
$ tt cartridge bench [BENCH_OPTION ...]
```

`tt cartridge bench` runs benchmarks for Tarantool.

### Options

`--url STRING`
: A Tarantool instance address (the default is `127.0.0.1:3301`).

`--user STRING`
: A username used to connect to the instance (the default is `guest`).

`--password STRING`
: A password used to connect to the instance.

`--connections INT`
: A number of concurrent connections (the default is `10`).

`--requests INT`
: A number of simultaneous requests per connection (the default is `10`).

`--duration INT`
: The duration of a benchmark test in seconds (the default is `10`).

`--keysize INT`
: The size of a key part of benchmark data in bytes (the default is `10`).

`--datasize INT`
: The size of a value part of benchmark data in bytes (the default is `20`).

`--insert INT`
: A percentage of inserts (the default is `100`).

`--select INT`
: A percentage of selects.

`--update INT`
: A percentage of updates.

`--fill INT`
: A number of records to pre-fill the space (the default is `1000000`).

### failover

```
$ tt cartridge failover COMMAND [COMMAND_OPTION ...]
```

`tt cartridge failover` manages an application failover. The following commands are available:

- `set`
- `setup`
- `status`
- `disable`

#### failover set

```
$ tt cartridge failover set MODE [FAILOVER_SET_OPTION ...]
```

Setup failover in the specified mode:

- `stateful`
- `eventual`
- `disabled`

Options:

- `--state-provider STRING`: A failover's state provider. Can be `stateboard` or `etcd2`. Used only in the `stateful` mode.
- `--params STRING`: Failover parameters specified in a JSON-formatted string, for example, `"{'fencing_timeout': 10', 'fencing_enabled': true}"`.
- `--provider-params STRING`: Failover provider parameters specified in a JSON-formatted string, for example, `"{'lock_delay': 14}"`.

#### failover setup

```
$ tt cartridge failover setup --file STRING
```

Setup failover with parameters described in a file. The failover configuration file defaults to `failover.yml`.

The `failover.yml` file might look as follows:

```
mode: stateful
state_provider: stateboard
stateboard_params:
    uri: localhost:4401
    password: passwd
failover_timeout: 15
```

#### failover status

```
$ tt cartridge failover status
```

Get the current failover status.

#### failover disable

```
$ tt cartridge failover disable
```

Disable failover.

#### Options

`--name STRING`
: An application name. Defaults to «package» in rockspec.

`--file STRING`
: A path to the file containing failover settings. Defaults to `failover.yml`.

### repair

```
$ tt cartridge repair COMMAND [REPAIR_OPTION ...]
```

`tt cartridge repair` repairs a running application. The following commands are available:

- `list-topology`
- `remove-instance`
- `set-advertise-uri`
- `set-leader`

### repair list-topology

```
$ tt cartridge repair list-topology [REPAIR_OPTION ...]
```

Get a summary of the current cluster topology.

### repair remove-instance

```
$ tt cartridge repair remove-instance UUID [REPAIR_OPTION ...]
```

Remove the instance with the specified UUID from the cluster. If the instance isn't found, raise an error.

### repair set-advertise-uri

```
$ tt cartridge repair set-advertise-uri INSTANCE-UUID NEW-URI [REPAIR_OPTION ...]
```

Change the instance's advertise URI. Raise an error if the instance isn't found or is expelled.

### repair set-leader

```
$ tt cartridge repair set-leader REPLICASET-UUID INSTANCE-UUID [REPAIR_OPTION ...]
```

Set the instance as the leader of the replica set. Raise an error in the following cases:

- There is no replica set or instance with that UUID.
- The instance doesn't belong to the replica set.
- The instance has been disabled or expelled.

#### Options

The following options work with any `repair` subcommand:

`--name`
: (Required) An application name.

`--data-dir`
: The directory containing the instances" working directories. Defaults to `/var/lib/tarantool`.

The following options work with any `repair` command, except `list-topology`:

`--run-dir`
: The directory where PID and socket files are stored. Defaults to `/var/run/tarantool`.

`--dry-run`
: Launch in dry-run mode: show changes but do not apply them.

`--reload`
: Enable instance configuration to reload after the patch.

### replicasets

```
$ tt cartridge replicasets COMMAND [COMMAND_OPTION ...]
```

`tt cartridge replicasets` manages an application's replica sets. The following commands are available:

- `setup`
- `save`
- `list`
- `join`
- `list-roles`
- `list-vshard-groups`
- `add-roles`
- `remove-roles`
- `set-weight`
- `set-failover-priority`
- `bootstrap-vshard`
- `expel`

#### replicasets setup

```
$ tt cartridge replicasets setup [--file FILEPATH] [--bootstrap-vshard]
```

Setup replica sets using a file.

Options:

- `--file`: A file with a replica set configuration. Defaults to `replicasets.yml`.
- `--bootstrap-vshard`: Bootstrap vshard upon setup.

#### replicasets save

```
$ tt cartridge replicasets save [--file FILEPATH]
```

Save the current replica set configuration to a file.

Options:

- `--file`: A file to save the configuration to. Defaults to `replicasets.yml`.

#### replicasets list

```
$ tt cartridge replicasets list [--replicaset STRING]
```

List the current cluster topology.

Options:

- `--replicaset STRING`: A replica set name.

#### replicasets join

```
$ tt cartridge replicasets join INSTANCE_NAME ... [--replicaset STRING]
```

Join the instance to a cluster. If a replica set with the specified alias isn't found in the cluster, it is created. Otherwise, instances are joined to an existing replica set.

Options:

- `--replicaset STRING`: A replica set name.

#### replicasets list-roles

```
$ tt cartridge replicasets list-roles
```

List the available roles.

#### replicasets list-vshard-groups

```
$ tt cartridge replicasets list-vshard-groups
```

List the available vshard groups.

#### replicasets add-roles

```
$ tt cartridge replicasets add-roles ROLE_NAME ... [--replicaset STRING] [--vshard-group␣
↪STRING]
```

Add roles to the replica set.

Options:

- `--replicaset STRING`: A replica set name.
- `--vshard-group STRING`: A vshard group for `vshard-storage` replica sets.

#### replicasets remove-roles

```
$ tt cartridge replicasets remove-roles ROLE_NAME ... [--replicaset STRING]
```

Remove roles from the replica set.

Options:

- `--replicaset STRING`: A replica set name.

#### replicasets set-weight

```
$ tt cartridge replicasets set-weight WEIGHT [--replicaset STRING]
```

Specify replica set weight.

Options:

- `--replicaset STRING`: A replica set name.

#### replicasets set-failover-priority

```
$ tt cartridge replicasets set-failover-priority INSTANCE_NAME ... [--replicaset STRING]
```

Configure replica set failover priority.

Options:

- `--replicaset STRING`: A replica set name.

#### replicasets bootstrap-vshard

```
$ tt cartridge replicasets bootstrap-vshard
```

Bootstrap vshard.

#### replicasets expel

```
$ tt cartridge replicasets expel INSTANCE_NAME ...
```

Expel one or more instances from the cluster.

### Printing the contents of .snap and .xlog files

```
$ tt cat FILE ... [OPTION ...]
```

`tt cat` prints the contents of *snapshot* (`.snap`) and *WAL* (`.xlog`) files to stdout. A single call of `tt cat` can print the contents of multiple files.

#### Options

`--format FORMAT`
: Output format: `yaml` (default), `json`, or `lua`.

`--from LSN`
: Show operations starting from the given LSN.

`--to LSN`
: Show operations up to the given LSN. Default: `18446744073709551615`.

`--replica ID`
: Filter the output by replica ID. Can be passed more than once.

  When calling `tt cat` with filters by LSN (`--from` and `--to` flags) and replica ID (`--replica`), remember that LSNs differ across replicas. Thus, if you pass more than one replica ID via `--from` or `--to`, the result may not reflect the actual sequence of operations.

`--space ID`
: Filter the output by space ID. Can be passed more than once.

`--show-system`
: Show the contents of system spaces.

### Examples

- Output contents of `00000000000000000000.xlog` WAL file in the YAML format:

```
$ tt cat 00000000000000000000.xlog
```

- Output operations on spaces with `space_id` 512 and 513 from the `00000000000000000012.snap` snapshot file in the JSON format:

```
$ tt cat 00000000000000000012.snap --space 512 --space 513 --format json
```

- Output operations on all spaces, including system spaces, from the `00000000000000000000.xlog` WAL file:

```
$ tt cat 00000000000000000000.xlog --show-system
```

- Output operations with LSNs between 100 and 500 on replica 1 from the `00000000000000000000.xlog` WAL file:

```
$ tt cat 00000000000000000000.xlog --from 100 --to 500 --replica 1
```

### Environment configuration

```
$ tt cfg COMMAND [OPTION ...]
```

`tt cfg` manages a `tt` environment *configuration*.

### Commands

| | |
|---|---|
| `dump` | Print a `tt` environment configuration.<br>Options:<br>• `-r`, `--raw`: Print a raw content of the `tt.yaml` configuration file. |

### Examples

Print the current `tt` environment configuration:

```
$ tt cfg dump
```

### Checking an application file

```
$ tt check {FILEPATH | APPLICATION[:APP_INSTANCE]}
```

`tt check` checks the syntax correctness of Lua files within Tarantool applications or separate Lua scripts. The files must be stored inside the `instances_enabled` directory specified in the *tt configuration file*.

### Examples

To check all Lua files in an application directory at once, specify the directory name:

```
$ tt check app
```

To check a single Lua file from an application directory, add the path to this file:

```
$ tt check app/router
# or
$ tt check app/router.lua
```

**Примечание:** The `.lua` extension can be omitted.

### Cleaning instance files

```
$ tt clean APPLICATION[:APP_INSTANCE] [OPTION ...]
```

`tt clean` cleans stored files of Tarantool instances: logs, snapshots, and other files. To avoid accidental deletion of files, `tt clean` shows the files it is going to delete and asks for confirmation.

When called without arguments, cleans files of all applications in the current environment.

### Options

`-f, --force`
: Clean files without confirmation.

### Examples

- Clean the files of all instances of the `app` application:

  ```
  $ tt clean app
  ```

- Clean the files of the `master` instance of the `app` application:

  ```
  $ tt clean app:master
  ```

### Managing cluster configurations

```
$ tt cluster COMMAND {APPLICATION[:APP_INSTANCE] | URI} [FILE] [OPTION ...]
```

`tt cluster` manages *YAML configurations* of Tarantool applications. This command works both with local configuration files in application directories and with centralized configuration storages (*etcd* or Tarantool-based).

`COMMAND` is one of the following:

- `publish`: publish a cluster configuration using an arbitrary YAML file as a source.
- `show`: print a cluster configuration.

### Managing local configurations

`tt cluster` can read and modify local cluster configurations stored in `config.yaml` files inside application directories.

To write a configuration to a local `config.yaml`, run `tt cluster publish` with two arguments:

- the application name.
- the path to a YAML file from which the configuration should be taken.

```
$ tt cluster publish myapp source.yaml
```

To print a local configuration from an application's `config.yaml`, run `tt cluster show` with the application name:

```
$ tt cluster show myapp
```

### Managing configurations in centralized storages

`tt cluster` can manage centralized cluster configurations in storages of both supported types: *etcd* or a Tarantool-based configuration storage.

To publish a configuration from a file to a centralized configuration storage, run `tt cluster publish` with a URI of this storage's instance as the target. For example, the command below publishes a configuration from `source.yaml` to a local etcd instance running on the default port 2379:

```
$ tt cluster publish "http://localhost:2379/myapp" source.yaml
```

A URI must include a prefix that is unique for the application. It can also include credentials and other connection parameters. Find the detailed description of the URI format in *URI format*.

To print a cluster configuration from a centralized storage, run `tt cluster show` with a storage URI including the prefix identifying the application. For example, to print `myapp`'s configuration from a local etcd storage:

```
$ tt cluster show "http://localhost:2379/myapp"
```

### Authentication

There are three ways to pass the credentials for connecting to the centralized configuration storage. They all apply to both etcd and Tarantool-based storages. The following list shows these ways ordered by precedence, from highest to lowest:

1. Credentials specified in the storage URI: `https://username:password@host:port/prefix`:

   ```
   $ tt cluster show "http://myuser:p4$$w0rD@localhost:2379/myapp"
   ```

2. `tt cluster` options `-u`/`--username` and `-p`/`--password`:

   ```
   $ tt cluster show "http://localhost:2379/myapp" -u myuser -p p4$$w0rD
   ```

3. Environment variables `TT_CLI_ETCD_USERNAME` and `TT_CLI_ETCD_PASSWORD`:

   ```
   $ export TT_CLI_ETCD_USERNAME=myuser
   $ export TT_CLI_ETCD_PASSWORD=p4$$w0rD
   $ tt cluster show "http://localhost:2379/myapp"
   ```

If connection encryption is enabled on the configuration storage, pass the required SSL parameters in the *URI arguments*.

### URI format

A URI of the cluster configuration storage has the following format:

```
http(s)://[username:password@]host:port[/prefix][?arguments]
```

- `username` and `password` define credentials for connecting to the configuration storage.
- `prefix` is a base path identifying a specific application in the storage.
- `arguments` defines connection parameters. The following arguments are available:
  - `name` – a name of an instance in the cluster configuration.
  - `key` – a target configuration key in the specified `prefix`.
  - `timeout` – a request timeout in seconds. Default: `3.0`.
  - `ssl_key_file` – a path to a private SSL key file.
  - `ssl_cert_file` – a path to an SSL certificate file.
  - `ssl_ca_file` – a path to a trusted certificate authorities (CA) file.
  - `ssl_ca_path` – a path to a trusted certificate authorities (CA) directory.
  - `ssl_ciphers` – a colon-separated (`:`) list of SSL cipher suites the connection can use (for Tarantool-based storage only).
  - `verify_host` – verify the certificate's name against the host. Default `true`.
  - `verify_peer` – verify the peer's SSL certificate. Default `true`.

### Managing configurations of specific instances

In addition to whole cluster configurations, `tt cluster` can manage configurations of specific instances within applications. In this case, it operates with YAML fragments that describe a single *instance configuration section*. For example, the following YAML file can be a source when publishing an instance configuration:

```
# instance_source.yaml
iproto:
  listen:
  - uri: 127.0.0.1:3311
```

To send an instance configuration to a local `config.yaml`, run `tt cluster publish` with the `application:instance` pair as the target argument:

```
$ tt cluster publish myapp:instance-002 instance_source.yaml
```

To send an instance configuration to a centralized configuration storage, specify the instance name in the `name` argument of the storage URI:

```
$ tt cluster publish "http://localhost:2379/myapp?name=instance-002" instance_source.yaml
```

`tt cluster show` can print configurations of specific cluster instances as well. To print an instance configuration from a local `config.yaml`, use the `application:instance` argument:

```
$ tt cluster show myapp:instance-002
```

To print an instance configuration from a centralized configuration storage, specify the instance name in the `name` argument of the URI:

```
$ tt cluster show "http://localhost:2379/myapp?name=instance-002"
```

### Configuration validation

`tt cluster` can validate configurations against the Tarantool configuration schema.

`tt cluster publish` automatically performs the validation and aborts in case of an error. To skip the validation, add the `--force` option:

```
$ tt cluster publish myapp source.yaml --force
```

To validate configurations when printing them with `tt cluster show`, enable the validation by adding the `--validate` option:

```
$ tt cluster show "http://localhost:2379/myapp" --validate
```

### Options

`-u, --username STRING`
: A username for connecting to the configuration storage.

  See also: *Authentication*.

`-p, --password STRING`
: A password for connecting to the configuration storage.

  See also: *Authentication*.

`--force`
: **Applicable to:** `publish`

  Skip validation when publishing. Default: `false` (validation is enabled).

`--validate`
: **Applicable to:** `show`

  Validate the printed configuration. Default: `false` (validation is disabled).

`--with-integrity-check STRING`

---

**Enterprise Edition**

This option is supported by the Enterprise Edition only.

---

**Applicable to:** `publish`

Generate hashes and signatures for integrity checks.

### Generating completion for tt

```
$ tt completion SHELL
```

`tt completion` generates tab-based completion for `tt` commands in the specified shell: `bash` or `zsh`.

### Examples

Generate `tt` completion for the current `bash` terminal:

```
$ . <(tt completion bash)
```

---

**Примечание:** You can add an execution of the completion script to a user's `.bashrc` file to make the completion work for this user in all their terminals.

---

### Connecting to a Tarantool instance

```
$ tt connect {URI|INSTANCE_NAME} [OPTION ...]
```

`tt connect` connects to a Tarantool instance by its URI or instance name specified in the current environment.

### Options

`-u USERNAME, --username USERNAME`
: A Tarantool user for connecting to the instance.

`-p PASSWORD, --password PASSWORD`
: The user's password.

`-f FILEPATH, --file FILEPATH`
: Connect and evaluate the script from a file.

  `-` – read the script from stdin.

`-i, --interactive`
: Enter the interactive mode after evaluating the script passed in `-f`/`--file`.

`-l LANGUAGE, --language LANGUAGE`
: The input language of the *tt interactive console*: `lua` (default) or `sql`.

`-x FORMAT, --outputformat FORMAT`
: The output format of the *tt interactive console*: `yaml` (default), `lua`, `table`, `ttable`.

`--sslcertfile FILEPATH`
: The path to an SSL certificate file for encrypted connections.

`--sslkeyfile FILEPATH`
: The path to a private SSL key file for encrypted connections.

`--sslcafile FILEPATH`
: The path to a trusted certificate authorities (CA) file for encrypted connections.

`--sslciphers STRING`
: The list of SSL cipher suites used for encrypted connections, separated by colons (`:`).

### Details

To connect to an instance, `tt` typically needs its URI – the host name or IP address and the port.

You can also connect to instances in the same `tt` environment (that is, those that use the same *configuration file* and Tarantool installation) by their instance names.

### Authentication

When connecting to an instance by its URI, `tt connect` establishes a remote connection for which authentication is required. Use one of the following ways to pass the username and the password:

- The `-u` (`--username`) and `-p` (`--password`) options:

```
$ tt connect 192.168.10.10:3301 -u myuser -p p4$$w0rD
```

- The connection string:

```
$ tt connect myuser:p4$$w0rD@192.168.10.10:3301
```

- Environment variables `TT_CLI_USERNAME` and `TT_CLI_PASSWORD`:

```
$ export TT_CLI_USERNAME=myuser
$ export TT_CLI_PASSWORD=p4$$w0rD
$ tt connect 192.168.10.10:3301
```

If no credentials are provided for a remote connection, the user is automatically `guest`.

**Примечание:** Local connections (by instance name instead of the URI) don't require authentication.

### Encrypted connection

To connect to instances that use SSL encryption, provide the SSL certificate and SSL key files in the `--sslcertfile` and `--sslkeyfile` options. If necessary, add other SSL parameters – `--sslcafile` and `--sslciphers`.

### Script evaluation

By default, `tt connect` opens an *interactive tt console*. Alternatively, you can open a connection to evaluate a Lua script from a file or stdin. To do this, pass the file path in the `-f` (`--file`) option or use `-f -` to take the script from stdin.

```
$ tt connect app -f test.lua
```

### Examples

- Connect to the `app` instance in the same environment:

```
$ tt connect app
```

- Connect to the `master` instance of the `app` application in the same environment:

```
$ tt connect app:master
```

- Connect to the `192.168.10.10` host on port `3301` with authentication:

```
$ tt connect 192.168.10.10:3301 -u myuser -p p4$$w0rD
```

- Connect to the `app` instance and evaluate the code from the `test.lua` file:

```
$ tt connect app -f test.lua
```

- Connect to the `app` instance and evaluate the code from stdin:

```
$ echo "function test() return 1 end" | tt connect app -f - # Create the test()␣
↪function
$ echo "test()" | tt connect app -f -                       # Call this function
```

### Manipulating Tarantool core dumps

```
$ tt coredump COMMAND [COMMAND_OPTION ...]
```

`tt coredump` provides commands for manipulating Tarantool core dumps.

To be able to investigate Tarantool crashes, make sure that core dumps are enabled on the host. Here is the *instruction on enabling core dumps on Unix systems*.

---

**Важно:** `tt coredump` does not support macOS.

---

### Commands

#### pack

```
$ tt coredump pack COREDUMP_FILE
```

Pack a Tarantool core dump and supporting data into a `tar.gz` archive. It includes:

- the Tarantool executable
- Tarantool version information
- OS information
- Shared libraries

Option: a path to a core dump file.

#### unpack

```
$ tt coredump unpack ARCHIVE
```

Unpack a Tarantool core dump created with `tt coredump pack` into a new directory.

Option: a path to a `tar.gz` archive packed by `tt coredump pack`.

### inspect

```
$ tt coredump inspect DIRECTORY
```

Inspect a Tarantool core dump directory with the GNU debugger (`gdb`) The directory being inspected must have the same structure as the core dump archive created by `tt coredump pack`.

---

**Примечание:** `tt coredump inspect` requires `gdb` installed on the host.

---

Option: a path to a directory with an unpacked core dump archive.

### Examples

- Pack a `tar.gz` file with a Tarantool core dump and supporting data:

  ```
  $ tt coredump pack name.core
  ```

- Unpack a `tar.gz` archive packed by `tt coredump pack`:

  ```
  $ tt coredump unpack tarantool-core-dump.tar.gz
  ```

- Inspect the unpacked core dump with `gdb`:

  ```
  $ tt coredump inspect tarantool-core-dump
  ```

### Creating an application from a template

```
$ tt create TEMPLATE_NAME [OPTION ...]
```

`tt create` creates a new Tarantool application from a template.

### Options

`-d PATH, --dst PATH`
: Path to the directory where the application will be created.

`-f, --force`
: Force rewrite the application directory if it already exists.

`--name NAME`
: Application name.

`-s, --non-interactive`
: Non-interactive mode.

`--var [VAR=VALUE ...]`
: Variable definition. Usage: `--var var_name=value`.

`--vars-file FILEPATH`
: Path to the file with variable definitions.

### Details

*Application templates* speed up the development of Tarantool applications by defining their initial structure and content. A template can include application code, configuration, build scripts, and other resources.

`tt` searches templates in the directories specified in the `templates` section of its *configuration file*.

### Template structure

Application templates are directories with files.

The main file of a template is its *manifest*. It defines how the applications are instantiated from this template.

A template manifest is a YAML file named `MANIFEST.yaml`. It can contain the following sections:

- `description` – the template description.
- `vars` – *template variables*.
- `pre-hook` and `post-hook` – paths to executables to run before and after the template instantiation.
- `include` – a list of files to keep in the application directory after instantiation. If this section is omitted, the application will contain all template files and directories.

All sections are optional.

Example:

```
description: Template description
vars:
  - prompt: User name
    name: user_name
    default: admin
    re: ^\w+$

  - prompt: Retry count
    default: "3"
    name: retry_count
    re: ^\d+$
pre-hook: ./hooks/pre-gen.sh
post-hook: ./hooks/post-gen.sh
include:
  - init.lua
  - instances.yml
```

Files and directories of a template are copied to the application directory according to the `include` section of the manifest (or its absence).

---

**Примечание:** Don't include the `.rocks` directory in application templates. To specify application dependencies, use the `.rockspec` files.

---

There is a special file type `*.tt.template`. The content of such files is adjusted for each application with the help of *template variables*. During the instantiation, the variables in these files are replaced with provided values and the `*.tt.template` extension is removed.

### Variables

Templates variables are replaced with their values provided upon the instantiation.

All templates have the `name` variable. Its value is taken from the `--name` option.

To add other variables, define them in the `vars` section of the template manifest. A variable can have the following attributes:

- `prompt`: a line of text inviting to enter the variable value in the interactive mode. Required.
- `name`: the variable name. Required.
- `default`: the default value. Optional.
- `re`: a regular expression that the value must match. Optional.

Example:

```
vars:
  - prompt: Cluster cookie
    name: cluster_cookie
    default: cookie
    re: ^\w+$
```

Variables can be used in all file names and the content of `*.tt template` files.

**Примечание:** Variables don't work in directory names.

To use a variable, enclose its name with a period in the beginning in double curly braces: `{{.var_name}}` (as in the Golang text templates syntax).

Examples:

- `init.lua.tt.template` file:

  ```
  local app_name = {{.name}}
  local login = {{.user_name}}
  ```

- A file name `{{.user_name}}.txt`

Variables receive their values during the template instantiation. By default, `tt create` asks you to provide the values interactively. You can use the `-s` (or `--non-interactive`) option to disable the interactive input. In this case, the values are searched in the following order:

- In the `--var` option. Pass a string of the `var=value` format after the `--var` option. You can pass multiple variables, each after a separate `--var` option:

  ```
  $ tt create template app --var user_name=admin
  ```

- In a file. Specify `var=value` pairs in a plain text file, each on a new line, and pass it as the value of the `--vars-file` option:

  ```
  $ tt create template app --vars-file variables.txt
  ```

  `variables.txt` can look like this:

```
user_name=admin
password=p4$$w0rd
version=2
```

If a variable isn't initialized in any of these ways, the default value from the manifest is used.

You can combine different ways of passing variables in a single call of `tt create`.

#### Application directory

By default, the application appears in the directory named after the provided application name (`--name` value).

To change the application location, use the `-dst` option.

#### Examples

- Create the application `app1` from the `simple_app` user-defined template in the current directory:

```
$ tt create simple_app --name app1
```

- Create the `app1` application in `/opt/tt/apps` from the built-in `vshard_cluster` template, force rewrite the application directory if it already exists. User interaction is disabled.

```
$ tt create vshard_cluster --name app1 -f --non-interactive -dst /opt/tt/apps/
```

### Interacting with the CRUD module

**Enterprise Edition**

This command is supported by the Enterprise Edition only.

```
$ tt crud COMMAND [COMMAND_OPTION ...]
```

`tt crud` enables the interaction with a cluster using the CRUD module. `COMMAND` is one of the following:

- `export`: export a cluster's data to a file. Learn more at *Exporting data*.
- `import`: import data from a file. Learn more at *Importing data*.

#### Exporting data

**Enterprise Edition**

This command is supported by the Enterprise Edition only.

```
$ tt [crud|tdg2] export URI SPACE:FILE ... [EXPORT_OPTION ...]
```

`tt [crud|tdg2] export` exports a space's data to a file. Three export commands cover the following cases:

- `tt export` exports data from a replica set using the *box.space* API.
- `tt crud export` exports data from a sharded cluster through a router using the CRUD module.
- `tt tdg2 export` exports data from a Tarantool Data Grid 2 cluster through its connector using TDG2 Repository API.

`tt [crud|tdg2] export` takes the following arguments:

- `URI`: The URI of a router instance if `crud` is used. Otherwise, it should specify the URI of a storage.
- `FILE`: The name of a file for storing exported data.
- `SPACE`: The name of a space from which data is exported.

**Примечание:** *Read access* to the space is required to export its data.

### Output format

`tt export` exports data in the following formats:

- `tt export` and `tt crud export`: CSV
- `tt tdg2 export`: JSON lines

### Limitations

Exporting isn't supported for the *interval* field type.

### Exporting with default settings

The command below exports data of the `customers` space to the `customers.csv` file:

```
$ tt crud export localhost:3301 customers:customers.csv
```

If the `customers` space has five fields (`id`, `bucket_id`, `firstname`, `lastname`, and `age`), the file with exported data might look like this:

```
1,477,Andrew,Fuller,38
2,401,Michael,Suyama,46
3,2804,Robert,King,33
# ...
```

If a tuple contains a `null` value, for example, `[1, 477, 'Andrew', null, 38]`, it is exported as an empty value:

```
1,477,Andrew,,38
```

### Exporting headers

To export data with a space's field names in the first row of the CSV file, use the `--header` option:

```
$ tt crud export localhost:3301 customers:customers.csv  \
                --header
```

In this case, field values start from the second row, for example:

```
id,bucket_id,firstname,lastname,age
1,477,Andrew,Fuller,38
2,401,Michael,Suyama,46
3,2804,Robert,King,33
# ...
```

### Exporting compound data

In the CSV format, `tt` exports empty values by default for fields containing compound data such as arrays or maps. To export compound values in a specific format, use the `--compound-value-format` option. For example, the command below exports compound values to CSV serialized in JSON:

```
$ tt crud export localhost:3301 customers:customers.csv  \
                --compound-value-format json
```

### Exporting from Tarantool Data Grid 2

---

**Примечание:** In the TDG2 data model, a **type** represents a Tarantool space, and an **object** of a type represents a tuple in the type's underlying space.

---

The command below exports data of the `customers` type from a TDG2 cluster to the `customers.jsonl` file:

```
$ tt tdg2 export localhost:3301 customers:customers.jsonl
```

If the `customers` type has four fields (`id`, `firstname`, `lastname`, and `age`), the file with exported data might look like this:

```
{"age":30,"first_name":"Samantha","id":1,"second_name":"Carter"}
{"age":41,"first_name":"Fay","id":2,"second_name":"Rivers"}
{"age":74,"first_name":"Milo","id":4,"second_name":"Walters"}
```

If an object contains a `null` value in a field, this field skipped:

```
{"age":13,"first_name":"Zachariah","id":3}
```

Object fields that contain maps with non-string keys are converted to maps with string keys.

TDG2 sets a limit on the number of objects transferred from each storage during a query execution (the hard-limits.returned TDG2 configuration parameter). If an export batch size (`--batch-size` parameter) is greater than this limit, it is possible that more than `hard-limits.returned` objects will be requested from one storage and export will fail. To make sure that `hard-limits.returned` is never exceeded during an export operation, set the export batch size less or equal to this limit.

For example, if your TDG2 cluster has a 1000 objects `hard-limits.returned` limit:

```
# tdg2 config.yaml
# ...
hard-limits.returned: 1000
```

Set the `tt tdg2 export` batch size less or equal to 1000:

```
$ tt tdg2 export localhost:3301 customers:customers.jsonl --batch-size=1000
```

### Options

`--batch-queue-size INT`
: The maximum number of tuple batches in a queue between a fetch and write threads (the default is 32).

  `tt` exports data using two threads:

  - A *fetch* thread makes requests and receives data from a Tarantool instance.
  - A *write* thread encodes received data and writes it to the output.

  The fetch thread uses a queue to pass received tuple batches to the write thread. If a queue is full, the fetch thread waits until the write thread takes a batch from the queue.

`--batch-size INT`
: The number of tuples to transfer per request. The default is:

  - 10000 for `tt export` and `tt crud export`.
  - 100 for `tt tdg2 export`.

  ---

  **Важно:** When using `tt tdg2 export`, make sure that the batch size does not exceed the `hard-limits.returned` TDG2 parameter value set on the cluster.

  ---

`--compound-value-format STRING`
: **Applicable to:** `tt export`, `tt crud export`

  A format used to export compound values like arrays or maps. By default, `tt` exports empty values for fields containing such values.

  Supported formats: `json`.

  See also: *Exporting compound data*.

`--header`
: **Applicable to:** `tt export`, `tt crud export`

  Add field names in the first row.

  See also: *Exporting headers*.

`--password STRING`
: A password used to connect to the instance.

`--readview`
: **Applicable to:** `tt export`, `tt crud export`

  Export data using a *read view*.

`--username STRING`
: A username for connecting to the instance.

### Displaying help for tt and its commands

```
$ tt help [COMMAND]
```

`tt help` displays help:

- for `tt` utility when called without a `COMMAND`.
- for a specified `tt` command.

### Examples

- Display `tt` help

  ```
  $ tt help
  ```

- Display help for the `start` command:

  ```
  $ tt help start
  ```

### Importing data

---

**Enterprise Edition**

This command is supported by the Enterprise Edition only.

---

```
$ tt [crud|tdg2] import URI FILE:SPACE [IMPORT_OPTION ...]
# or
$ tt [crud|tdg2] import URI :SPACE < FILE [IMPORT_OPTION ...]
```

`tt [crud|tdg] import` imports data from a file to a space. Three import commands cover the following cases:

- `tt import` imports data into a replica set through its master instance using the *box.space* API.
- `tt crud import` imports data into a sharded cluster through a router using the CRUD module.
- `tt tdg2 import` imports data into a Tarantool Data Grid 2 cluster through its router using the `repository.put` function of the TDG2 Repository API.

`tt [crud|tdg2] import` takes the following arguments:

- `URI`: The URI of a router instance if `crud` is used. Otherwise, it should specify the URI of a storage.
- `FILE`: The name of a file containing data to be imported.
- `SPACE`: The name of a space to which data is imported.

---

**Примечание:** *Write access* to the space and `execute` access to `universe` are required to import data.

---

#### Input file format

`tt import` imports data from the following formats:

- `tt import` and `tt crud import`: CSV
- `tt tdg2 import`: JSON lines

#### Limitations

Importing isn't supported for the *interval* field type.

### Matching of input and space fields

#### Automatic matching

Suppose that you have the `customers.csv` file with a header containing field names in the first row:

```
id,firstname,lastname,age
1,Andrew,Fuller,38
2,Michael,Suyama,46
3,Robert,King,33
# ...
```

If the target `customers` space has fields with the same names, you can import data using the `--header` and `--match` options specified as follows:

```
$ tt crud import localhost:3301 customers.csv:customers \
                 --header \
                 --match=header
```

In this case, fields in the input file and the target space are matched automatically. You can also match fields *manually* if field names in the input file and the target space differ. Note that if you're importing data into a cluster, you don't need to specify the `bucket_id` field. The CRUD module generates `bucket_id` values automatically.

#### Manual matching

The `--match` option enables importing data by matching field names in the input file and the target space manually. Suppose that you have the following `customers.csv` file with four fields:

```
customer_id,name,surname,customer_age
1,Andrew,Fuller,38
2,Michael,Suyama,46
3,Robert,King,33
# ...
```

If the target `customers` space has the `id`, `firstname`, `lastname`, and `age` fields, you can configure mapping as follows:

```
$ tt crud import localhost:3301 customers.csv:customers \
                --header \
                --match "id=customer_id;firstname=name;lastname=surname;age=customer_age
↪"
```

Similarly, you can configure mapping using numeric field positions in the input file:

```
$ tt crud import localhost:3301 customers.csv:customers \
                --header \
                --match "id=1;firstname=2;lastname=3;age=4"
```

Below are the rules if some fields are missing in input data or space:

- If a space has fields that are not specified in input data, `tt [crud] import` tries to insert `null` values.
- If input data contains fields missing in a target space, these fields are ignored.

#### Handling duplicate primary key errors

The `--on-exist` option enables you to control data import when a duplicate primary key error occurs. In the example below, values already existing in the space are replaced with new ones:

```
$ tt crud import localhost:3301 customers.csv:customers \
                --on-exist replace
```

#### Handling parsing errors

To skip rows whose data cannot be parsed correctly, use the `--on-error` option as follows:

```
$ tt crud import localhost:3301 customers.csv:customers \
                --on-error skip
```

#### Importing into Tarantool Data Grid 2

---

**Примечание:** In the TDG2 data model, a **type** represents a Tarantool space, and an **object** of a type represents a tuple in the type's underlying space.

---

The command below imports objects of the `customers` type into a TDG2 cluster. The objects are described in the `customers.jsonl` file.

```
$ tt tdg2 import localhost:3301 customers.jsonl:customers
```

The input file can look like this:

```
{"age":30,"first_name":"Samantha","id":1,"second_name":"Carter"}
{"age":41,"first_name":"Fay","id":2,"second_name":"Rivers"}
{"age":74,"first_name":"Milo","id":4,"second_name":"Walters"}
```

**Примечание:** Since JSON describes objects in maps with string keys, there is no way to import a field value that is a map with a non-string key.

In case of an error during TDG2 import, `tt tdg2 import` rolls back the changes made *within the current batch* on the *storage where the error has happened* (per-storage rollback) and reports an error. On other storages, objects from the same batch can be successfully imported. So, the rollback process of `tt tdg2 import` is the same as the one of `tt crud import` with the `--rollback-on-error` option.

Since object batches can be imported partially (per-storage rollback), the absence of error matching complicates the debugging in case of errors. To minimize this effect, the default batch size (`--batch-size`) for `tt tdg2 import` is 1. This makes the debugging straightforward: you always know which object caused the error. On the other hand, this decreases the performance in comparison to import in larger batches.

If you increase the batch size, `tt` informs you about the possible issues and asks for an explicit confirmation to proceed. To automatically confirm a batch import operation, add the `--force` option:

```
$ tt tdg2 import localhost:3301 customers.jsonl:customers \
                --batch-size=100 \
                --force
```

### Options

`--batch-size INT`
: **Applicable to:** `tt crud import`, `tt tdg2 import`

  The number of tuples to transfer per request. The default is:

  - 100 for `tt crud import`.
  - 1 for `tt tdg2 import`. See *Importing into Tarantool Data Grid 2* for details.

`--dec-sep STRING`
: **Applicable to:** `tt import`, `tt crud import`

  The string of symbols that defines decimal separators for numeric data (the default is `.,`).

  **Примечание:** Symbols specified in this option cannot intersect with `--th-sep`.

`--delimiter STRING`
: **Applicable to:** `tt import`, `tt crud import`

  A symbol that defines a field value delimiter. For CSV, the default delimiter is a comma (`,`). To use a tab character as a delimiter, set this value as `tab`:

  ```
  $ tt crud import localhost:3301 customers.csv:customers \
                  --delimiter tab
  ```

  **Примечание:** A delimiter cannot be `\r`, `\n`, or the Unicode replacement character (`U+FFFD`).

`--error STRING`
: The name of a file containing rows that are not imported (the default is `error`).

  See also: *Handling parsing errors*.

`--force`
**Applicable to:** `tt tdg2 import`

Automatically confirm importing into TDG2 with `--batch-size` greater than one.

`--format STRING`
A format of input data.

Supported formats: `csv`.

`--header`
**Applicable to:** `tt import`, `tt crud import`

Process the first line as a header containing field names. In this case, field values start from the second line.

See also: *Matching of input and space fields*.

`--log STRING`
The name of a log file containing information about import errors (the default is `import`). If the log file already exists, new data is written to this file.

`--match STRING`
**Applicable to:** `tt import`, `tt crud import`

Configure matching between field names in the input file and the target space.

See also: *Matching of input and space fields*.

`--null STRING`
**Applicable to:** `tt import`, `tt crud import`

A value to be interpreted as `null` when importing data. By default, an empty value is interpreted as `null`. For example, a tuple imported from the following row ...

```
1,477,Andrew,,38
```

... should look as follows: `[1, 477, 'Andrew', null, 38]`.

`--on-error STRING`
An action performed if a row to be imported cannot be parsed correctly. Possible values:

- `stop`: stop importing data.
- `skip`: skip rows whose data cannot be parsed correctly.

Duplicate primary key errors are handled using the `--on-exist` option.

See also: *Handling parsing errors*.

`--on-exist STRING`
An action performed if a duplicate primary key error occurs. Possible values:

- `stop`: stop importing data.
- `skip`: skip existing values when importing.
- `replace`: replace existing values when importing.

Other errors are handled using the `--on-error` option.

See also: *Handling duplicate primary key errors*.

`--password STRING`
A password used to connect to the instance.

`--progress STRING`
: The name of a progress file that stores the following information:

  - The positions of lines that were not imported at the last launch.
  - The last position that was processed at the last launch.

  If a file with the specified name exists, it is taken into account when importing data. `tt import` tries to insert lines that were not imported and then continues importing from the last position.

  At each launch, the content of a progress file with the specified name is overwritten. If the file with the specified name does not exist, a progress file is created with the results of this run.

  ---

  **Примечание:** If the option is not set, then this mechanism is not used.

  ---

`--quote STRING`
: **Applicable to:** `tt import`, `tt crud import`

  A symbol that defines a quote. For CSV, double quotes are used by default (`"`). The double symbol of this option acts as the escaping symbol within input data.

`-success STRING`
: The name of a file with rows that were imported (the default is `success`). Overwrites the file if it already exists.

`--th-sep STRING`
: **Applicable to:** `tt import`, `tt crud import`

  The string of symbols that define thousand separators for numeric data. The default value includes a space and a backtick `` ` ``. This means that `1 000 000` and `1`000`000` are both imported as `1000000`.

  ---

  **Примечание:** Symbols specified in this option cannot intersect with `--dec-sep`.

  ---

`--username STRING`
: A username for connecting to the instance.

`--rollback-on-error`
: **Applicable to:** `tt crud import`

  Specify whether any operation failed on a storage leads to rollback of a batch import on this storage.

  ---

  **Примечание:** `tt tdg2 import` always works as if `--rollback-on-error` is `true`.

  ---

### Creating a tt environment

```
$ tt init
```

`tt init` creates a `tt` environment in the current directory. This includes:

- Setting up directories for working files: binaries, templates, and so on.
- Creating a corresponding `tt.yaml` *configuration file*.

### Details

**Важно:** The Tarantool Cartridge framework is deprecated and is not compatible with Tarantool 3.0 and later. This command is added for backward compatibility with earlier versions.

`tt init` checks the existence of configuration files for Cartridge (`cartridge.yml`) or the `tarantoolctl` utility (`.tarantoolctl`) in the current directory. If such files are found, `tt` generates an environment that uses the same directories:

- `cartridge.yml` – the directories specified in the file.
- `.tarantoolctl` – the directories specified in the `default_cfg` table.

  **Примечание:** `init` is the only `tt` command that invokes `.tarantoolctl` files. Thus, variables defined in this script will not be available in applications launched by a `tt start` call.

If there is no `cartridge.yml` or `.tarantoolctl` files in the current directory, `tt init` creates a default environment in it. This includes creating the following directories and files:

- `bin` – the directory for storing binary files.
- `include` – the directory for storing header files.
- `distfiles` – the directory for storing installation files.
- `instances.enabled` – the directory for storing running applications or symlinks.
- `modules` – the directory for storing external modules.
- `tt.yaml` – the configuration file.
- `templates` – the directory for storing application templates.

### Example

Create a `tt` environment in the current directory:

```
$ tt init
```

### Installing Tarantool software

```
$ tt install PROGRAM_NAME [VERSION|COMMIT_HASH|PR_ID] [OPTION ...]
```

`tt install` installs the latest or an explicitly specified version of Tarantool or `tt`. The possible values of `PROGRAM_NAME` are:

- `tarantool`: Install Tarantool Community Edition.
- `tarantool-dev`: Install Tarantool from a local build directory.
- `tarantool-ee`: Install Tarantool Enterprise Edition.
- `tt`: Install the `tt` command-line utility.

**Примечание:** For `tarantool-ee`, account credentials are required. Specify them in a file (see the *ee section* of the configuration file) or provide them interactively.

Additionally, `tt install` can build open source programs `tarantool` and `tt` from a specific commit or a pull request on their GitHub repositories.

To uninstall a Tarantool or `tt` version, use *tt uninstall*.

### Options

`--dynamic`
**Applicable to:** `tarantool`, `tarantool-ee`

Use dynamic linking for building Tarantool.

`-f, --force`
Skip dependency check before installation.

`--local-repo`
Install a program from the local repository, which is specified in the *repo section* of the `tt` configuration file.

`--no-clean`
Don't delete temporary files.

`--reinstall`
Reinstall a previously installed program.

`--use-docker`
**Applicable to:** `tarantool`, `tarantool-ee`

Build Tarantool in an Ubuntu 18.04 Docker container.

### Details

When called without an explicitly specified version, `tt install` installs the latest available version. To check versions available for installation, use *tt search*.

By default, available versions of Tarantool Community Edition and `tt` are taken from their git repositories. Their installation includes building from sources, which requires some tools and dependencies, such as a C compiler. Make sure they are available in the system.

Tarantool Enterprise Edition is installed from prebuilt packages.

### Authentication

To install Tarantool EE using `tt install`, you need to provide access credentials for Tarantool customer zone. Use one of the following ways to pass the username and the password:

- A text file specified in the `ee.credentials_path` parameter of the *tt enviromnment configuration*:

```
# tt.yaml
# ...
ee:
  credential_path: cred.txt
```

`cred.txt` should contain a username and a password on separate lines:

```
myuser@tarantool.io
p4$$w0rD
```

- Environment variables `TT_CLI_EE_USERNAME` and `TT_CLI_EE_PASSWORD`:

```
$ export TT_CLI_EE_USERNAME=myuser@tarantool.io
$ export TT_CLI_EE_PASSWORD=p4$$w0rD
$ tt install tarantool-ee
```

### Development versions

`tt install` can be used to build custom Tarantool and `tt` versions for development purposes from commits and pull requests on their GitHub repositories.

To build Tarantool or `tt` from a specific commit on their GitHub repository, pass the commit hash (7 or more characters) after the program name. If you want to use a PR as a source, provide a `pr/<PR_ID>` argument:

```
$ tt install tarantool 03c184d
$ tt install tt pr/50
```

If you *build Tarantool from sources*, you can install local builds to the current `tt` environment by running `tt install` with the `tarantool-dev` program name and the path to the build:

```
$ tt install tarantool-dev ~/src/tarantool/build
```

### Local repositories

You can also set up a local repository with installation files you need. To use it, specify its location in the *repo section* of the `tt` configuration file and run `tt install` with the `--local-repo` flag.

### Example

- Install the latest available version of Tarantool CE:

```
$ tt install tarantool
```

- Install Tarantool 2.11.1 from the local repository:

```
$ tt install tarantool 2.11.1 --local-repo
```

- Reinstall Tarantool 2.10.8:

```
$ tt install tarantool 2.10.8 --reinstall
```

- Install Tarantool from a PR #1234 on the tarantool/tarantool GitHub repository:

```
$ tt install tarantool pr/1234
```

- Install `tt` from a commit with a hash `40e696e` on the tarantool/tt GitHub repository:

```
$ tt install tt 40e696e
```

- Install Tarantool *built from sources*:

```
$ tt install tarantool-dev ~/src/tarantool/build
```

### Listing enabled applications

```
$ tt instances
```

`tt instances` shows the list of enabled applications and their instances in the current environment.

---

**Примечание:** *Enabled* applications are applications that are stored inside the `instances_enabled` directory specified in the *tt configuration file*. They can be either running or not. To check if an application is running, use *tt status*.

---

### Example

- Show the list of enabled applications and their instances:

```
$ tt instances
```

### Rotating instance logs

```
$ tt logrotate APPLICATION[:APP_INSTANCE]
```

`tt logrotate` rotates logs of a Tarantool application or specific instances, and the `tt` log. For example, you need to call this function to continue logging after a log rotation program renames or moves instances" logs. Learn more about *rotating logs*.

Calling `tt logrotate` on an application has the same effect as executing the built-in *log.rotate()* function on all its instances.

### Examples

Rotate logs of the `app` application's instances:

```
$ tt logrotate app
```

### Packaging the application

```
$ tt pack TYPE [OPTION ...] ..
```

`tt pack` packages an application into a distributable bundle of the specified `TYPE`:

- `tgz`: create a `.tgz` archive.
- `deb`: create a DEB package.
- `rpm`: create an RPM package.

The command below creates a DEB package for an application:

```
$ tt pack deb
```

This command generates a `.deb` file whose name depends on the environment directory name and the operating system architecture, for example, `test-env_0.1.0.0-1_x86_64.deb`. You can also pass various *options* to the `tt pack` command to adjust generation properties, for example, customize a bundle name, choose which artifacts should be included, specify the required application dependencies.

### Options

`--all`
: Include all artifacts in a bundle. In this case, a bundle might include snapshots, WAL files, and logs.

`--app-list APPLICATIONS`
: Specify the applications included in a bundle.

  **Example**

  ```
  $ tt pack tgz --app-list app1,app3
  ```

`--cartridge-compat`
: **Applicable to:** `tgz`

  Package a Cartridge CLI-compatible archive.

  ---

  **Важно:** The Tarantool Cartridge framework is deprecated and is not compatible with Tarantool 3.0 and later. This command is added for backward compatibility with earlier versions.

  ---

`--deps STRINGS`
: **Applicable to:** `deb`, `rpm`

  Specify dependencies included in RPM and DEB packages.

  **Example**

  ```
  $ tt pack deb --deps 'wget,make>0.1.0,unzip>1,unzip<=7'
  ```

`--deps-file STRING`
: **Applicable to:** `deb`, `rpm`

  Specify the path to a file containing dependencies included in RPM and DEB packages. For example, the `package-deps.txt` file below contains several dependencies and their versions:

```
unzip==6.0
neofetch>=6,<7
gcc>8
```

If this file is placed in the current directory, a `tt pack` command might look like this:

```
$ tt pack deb --deps-file package-deps.txt
```

`--filename`
: Specify a bundle name.

**Example**

```
$ tt pack tgz --filename sample-app.tar.gz
```

`--name PACKAGE_NAME`
: Specify a package name.

**Example**

```
$ tt pack tgz --name sample-app --version 1.0.1
```

`--preinst`
: **Applicable to:** `deb`, `rpm`

Specify the path to a pre-install script for RPM and DEB packages.

**Example**

```
$ tt pack deb --preinst pre.sh
```

`--postinst`
: **Applicable to:** `deb`, `rpm`

Specify the path to a post-install script for RPM and DEB packages.

**Example**

```
$ tt pack deb --postinst post.sh
```

`--tarantool-version`
: Specify a Tarantool version for packaging in a Docker container. For use with `--use-docker` only.

`--use-docker`
: Build a package in an Ubuntu 18.04 Docker container. To specify a Tarantool version to use in the container, add the `--tarantool-version` option.

Before executing `tt pack` with this option, make sure Docker is running.

`--version PACKAGE_VERSION`
: Specify a package version.

**Example**

```
$ tt pack tgz --name sample-app --version 1.0.1
```

`--with-binaries`
: Include Tarantool and `tt` binaries in a bundle.

`--without-binaries`
: Don't include Tarantool and `tt` binaries in a bundle.

`--without-modules`
: Don't include external modules in a bundle.

### Playing the contents of .snap and .xlog files to a Tarantool instance

```
$ tt play URI FILE ... [OPTION ...]
```

`tt play` plays the contents of *snapshot* (`.snap`) and *WAL* (`.xlog`) files to another Tarantool instance. A single call of `tt play` can play multiple files.

### Options

`-u USERNAME, --username USERNAME`
: A Tarantool user for connecting to the instance.

`-p PASSWORD, --password PASSWORD`
: The user's password.

`--from LSN`
: Play operations starting from the given LSN.

`--to LSN`
: Play operations up to the given LSN. Default: `18446744073709551615`.

`--replica ID`
: Filter the operations by replica ID. Can be passed more than once.

  When calling `tt cat` with filters by LSN (`--from` and `--to` flags) and replica ID (`--replica`), remember that LSNs differ across replicas. Thus, if you pass more than one replica ID via `--from` or `--to`, the result may not reflect the actual sequence of operations.

`--space ID`
: Filter the output by space ID. Can be passed more than once.

`--show-system`
: Show the operations on system spaces.

### Details

`tt play` plays operations from `.xlog` and `.snap` files to the destination instance one by one. All data changes happen the same way as if they were performed on this instance. This means that:

- All affected spaces must exist on the destination instance. They must have the same structure and `space_id` as on the instance that created the snapshot or WAL file.

  To play a snapshot or a WAL to a clean instance, include the operations on system spaces by adding the `--show-system` flag. With this flag, `tt` plays the operations that create and configure user-defined spaces.

- The operations" LSNs change unless you play all operations that took place since the instance startup.
- Replica IDs change in accordance with the destination instance configuration.

### Authentication

Use one of the following ways to pass the username and the password when connecting to the instance:

- The `-u` (`--username`) and `-p` (`--password`) options:

```
$ tt play 192.168.10.10:3301 00000000000000000000.xlog -u myuser -p p4$$w0rD
```

- The connection string:

```
$ tt play myuser:p4$$w0rD@192.168.10.10:3301 00000000000000000000.xlog
```

- Environment variables `TT_CLI_USERNAME` and `TT_CLI_PASSWORD`:

```
$ export TT_CLI_USERNAME=myuser
$ export TT_CLI_PASSWORD=p4$$w0rD
$ tt play 192.168.10.10:3301 00000000000000000000.xlog
```

### Examples

- Play the contents of `00000000000000000000.xlog` to the instance on `192.168.10.10:3301`:

```
$ tt play 192.168.10.10:3301 00000000000000000000.xlog
```

- Play operations on spaces with `space_id` 512 and 513 from the `00000000000000000012.snap` snapshot file:

```
$ tt play 192.168.10.10:3301 00000000000000000012.snap --space 512 --space 513
```

- Play the contents of `00000000000000000000.xlog` including operations on system spaces:

```
$ tt play 192.168.10.10:3301 00000000000000000000.xlog --show-system
```

### Restarting a Tarantool instance

```
$ tt restart APPLICATION[:APP_INSTANCE] [OPTION ...]
```

`tt restart` restarts the specified running Tarantool instance. A `tt restart` call is equivalent to consecutive calls of *tt stop* and *tt start*.

When called without arguments, restarts all running applications in the current environment.

See also: *Starting Tarantool applications*, *Stopping a Tarantool instance*, *Checking instance status*.

### Options

`-y, --yes`
: Automatic «Yes» to confirmation prompt.

### Examples

- Restart all instances of the application stored in the `app` directory inside `instances_enabled` in accordance with the *instances configuration*:

```
$ tt restart app
```

---

**Примечание:** This call starts all application instances specified in its `instances.yml`, even those that were not running before the call.

---

- Restart only the `master` instance of the `app` application with automatic confirmation:

```
$ tt restart app:master -y
```

## Using the LuaRocks package manager

```
$ tt rocks [OPTION ...] [VAR=VALUE] COMMAND [ARGUMENT]
```

`tt rocks` provides means to manage Lua modules (rocks) via the LuaRocks package manager. `tt` uses its own LuaRocks installation connected to the Tarantool rocks repository.

Below are lists of supported LuaRocks flags and commands. For detailed information on their usage, refer to LuaRocks documentation.

### Options

`--dev`
: Enable the sub-repositories in rocks servers for rockspecs of in-development versions.

`--server=SERVER`
: Fetch rocks/rockspecs from this server (takes priority over config file).

`--only-server=SERVER`
: Fetch rocks/rockspecs from this server only (overrides any entries in the config file).

`--only-sources=URL`
: Restrict downloads to paths matching the given URL.

`--lua-dir=PREFIX`
: Specify which Lua installation to use

`--lua-version=VERSION`
: Specify which Lua version to use.

`--tree=TREE`
: Specify which tree to operate on.

`--local`
: Use the tree in the user's home directory. Call `tt rocks help path` to learn how to enable it.

`--global`
: Use the system tree when `local_by_default` is `true`.

`--verbose`
: Display verbose output for the command executed.

`--timeout=SECONDS`
: Timeout on network operations, in seconds. `0` means no timeout (wait forever). Default: `30`.

### Commands

| | |
|---|---|
| `admin` | Use the luarocks-admin tool |
| `build` | Build and compile a rock |
| `config` | Query information about the LuaRocks configuration |
| `doc` | Show documentation for an installed rock |
| `download` | Download a specific rock file from a rocks server |
| `help` | Help on commands. Type `tt rocks help <command>` for more |
| `init` | Initialize a directory for a Lua project using LuaRocks |
| `install` | Install a rock |
| `lint` | Check syntax of a rockspec |
| `list` | List the currently installed rocks |
| `make` | Compile package in the current directory using a rockspec |
| `make_manifest` | Compile a manifest file for a repository |
| `new_version` | Auto-write a rockspec for a new version of a rock |
| `pack` | Create a rock, packing sources or binaries |
| `purge` | Remove all installed rocks from a tree |
| `remove` | Uninstall a rock |
| `search` | Query the LuaRocks servers |
| `show` | Show information about an installed rock |
| `test` | Run the test suite in the current directory |
| `unpack` | Unpack the contents of a rock |
| `which` | Tell which file corresponds to a given module name |
| `write_rockspec` | Write a template for a rockspec file |

### Examples

- Install the rock `queue` from the Tarantool rocks repository:

```
$ tt rocks install queue
```

- Search for the rock `queue` in **both** the Tarantool rocks repository and the default LuaRocks repository:

```
$ tt rocks search queue --server='https://luarocks.org'
```

- List the documentation files for the installed rock `queue`:

```
$ tt rocks doc queue --list
```

  Without the `--list` flag, this command displays documentation in the user's default browser.

- Create a `*.rock` file from the installed rock `queue`:

```
$ tt rocks pack queue
```

- Unpack a `*.rock` file:

```
$ tt rocks unpack queue-scm-1.all.rock
```

- Remove the installed rock `queue`:

```
$ tt rocks remove queue
```

### Running code in a Tarantool instance

```
$ tt run [SCRIPT|-e EXPR] [OPTION ...]
```

`tt run` executes Lua code in a new Tarantool instance.

#### Options

`-e EXPR, --evaluate EXPR`
: Execute the specified expression in a Tarantool instance.

`-l LIB_NAME, --library LIB_NAME`
: Require the specified library.

`-i, --interactive`
: Enter the interactive mode after the script execution.

`-v, --version`
: Print the Tarantool version that is used for script execution.

#### Details

`tt run` executes arbitrary Lua code in a Tarantool instance. The code can be provided either in a Lua file, or in a string passed after the `-e`/`--evaluate` flag. When called without arguments or flags, `tt run` opens the Tarantool console.

If libraries are required for execution, pass their names after the `-l`/`--library` flag.

By default, a Tarantool instance started by `tt run` shuts down after code execution completes. To leave this instance running and continue working in its console, add the `-i`/`--interactive` flag.

#### Examples

- Execute the `app.lua` file in a Tarantool instance:

```
$ tt run app.lua
```

- Execute an expression in a Tarantool instance:

```
$ tt run -e "print('hi there')"
```

- Execute the `app.lua` file in a Tarantool instance and leave it running:

```
$ tt run -i app.lua
```

### Listing available Tarantool versions

```
$ tt search PROGRAM_NAME [OPTION ...]
```

`tt search` lists versions of Tarantool and `tt` that are available for installation. The possible values of `PROGRAM_NAME` are:

- `tarantool`
- `tarantool-ee`
- `tt`

---

**Примечание:** For `tarantool-ee`, account credentials are required. Specify them in a file (see the *ee section* of the configuration file) or provide interactively.

---

### Options

`--debug`
: **Applicable to:** `tarantool-ee`

  Search for debug builds of Tarantool Enterprise Edition's SDK.

`--local-repo`
: Search in the local repository, which is specified in the *repo section* of the `tt` configuration file.

`--version VERSION`
: **Applicable to:** `tarantool-ee`

  Tarantool Enterprise version.

### Example

- List available Tarantool versions:

```
$ tt search tarantool
```

- List available `tt` versions from the local repository:

```
$ tt search --local-repo tt
```

### Starting Tarantool applications

```
$ tt start APPLICATION[:APP_INSTANCE]
```

`tt start` starts Tarantool applications. The application files must be stored inside the `instances_enabled` directory specified in the *tt configuration file*. For detailed instructions on preparing and running Tarantool applications, see *Application environment* and *Starting and stopping instances*.

When called without arguments, starts all enabled applications in the current environment.

See also: *Stopping a Tarantool instance*, *Restarting a Tarantool instance*, *Checking instance status*.

### Application layout

`tt start` can start entire Tarantool clusters based on their YAML configurations. A cluster application directory inside `instances_enabled` must contain the following files:

- `config.yaml` – a *YAML configuration* that defines the cluster topology and settings. It can either contain an explicit configuration in the YAML format or point to a centralized configuration storage (for Enterprise Edition).
- `instances.yml` – a file that defines the list of cluster instances to run in the current environment.
- (Optionally) `*.lua` files with code to load and run in the cluster.

For more information about Tarantool application layout, see *Application environment*.

---

**Примечание:** `tt` also supports Tarantool applications with *configuration in code*, which is considered a legacy approach since Tarantool 3.0. For information about using `tt` with such applications, refer to the Tarantool 2.11 documentation.

---

### Running in the background

`tt start` runs Tarantool applications in the background and uses its own watchdog process for status checks (*tt status*) and application stopping (*tt stop*).

---

**Важно:** Do not switch on the background mode using the cluster configuration (`process.background: true` in the YAML configuration) or code (`box.cfg.background = true`) in applications that you run with `tt`. If you start such an application with `tt start`, `tt` won't be able to check the application status or stop it using the corresponding commands.

---

### Examples

- Start instances of the application stored in the `app` directory inside `instances_enabled` in accordance with its `instances.yml`:

  ```
  $ tt start app
  ```

- Start the `router` instance of the `app` application:

```
$ tt start app:router
```

### Checking instance status

```
$ tt status APPLICATION[:APP_INSTANCE]
```

`tt status` prints the current status of Tarantool applications and instances in the current environment. When called without arguments, prints the status of all enabled applications in the current environment.

### Examples

- Check the status of all instances of the `app` application:

  ```
  $ tt status app
  ```

- Check the status of the `replica` instance of the `app` application:

  ```
  $ tt status app:replica
  ```

### Stopping a Tarantool instance

```
$ tt stop APPLICATION[:APP_INSTANCE]
```

`tt stop` stops the specified running Tarantool applications or instances. When called without arguments, stops all running applications in the current environment.

See also: *Starting Tarantool applications*, *Restarting a Tarantool instance*, *Checking instance status*.

### Examples

- Stop all instances of the `app` application:

  ```
  $ tt stop app
  ```

- Stop the `replica` instance of the `app` application:

  ```
  $ tt stop app:replica
  ```

### Interacting with the Tarantool Data Grid 2

**Enterprise Edition**

This command is supported by the Enterprise Edition only.

```
$ tt tdg2 COMMAND [COMMAND_OPTION ...]
```

`tt tdg2` enables the interaction with Tarantool Data Grid 2 clusters. `COMMAND` is one of the following:

- `export`: export a TDG2 cluster's data to a file. Learn more at *Exporting data*.
- `import`: import data to a TDG2 cluster from a file. Learn more at *Importing data*.

### Uninstalling Tarantool software

```
$ tt uninstall PROGRAM_NAME [VERSION]
```

`tt uninstall` uninstalls a previously *installed* Tarantool version.

### Example

Uninstall Tarantool 2.10.4:

```
$ tt uninstall tarantool 2.10.4
```

### Displaying the tt version

```
$ tt version
```

`tt version` shows the version of the `tt` utility being used.

### Extending the tt functionality

The `tt` utility implements a modular architecture: its *commands* are, in fact, separate modules. When you run `tt` with a command, the corresponding module is executed with the given arguments.

The modular architecture enables the option to extend the `tt` functionality with **external modules** (as opposed to **internal modules** that implement built-in commands). Simply said, you can write any code you want to execute from `tt`, pack it into an executable, and run it with a `tt` command:

```
tt my-module-name my-args
```

The name of the command that executes a module is the same as the name of the module's executable.

### Module description and help

Executables that implement external `tt` modules must have two flags:

- `--description` – print a short description of the module. The description is shown alongside the command in the `tt` help.
- `--help` – display help. The help message is shown when `tt help <module_name>` is called.

### Location

External modules must be located in the modules directory specified in the *configuration file*:

```
tt:
  modules:
    directory: path/to/modules/dir
```

To check if a module is available in `tt`, call `tt help`. It will show the available external modules in the `EXTERNAL COMMANDS` section together with their descriptions.

### Overloading built-in commands

External modules can overload built-in `tt` commands. If you want to change the behavior of a built-in command, create an external module with the same name and your own implementation.

When `tt` sees two modules – an external and an internal one – with the same name, it will use the external module by default.

For example, if you want `tt` to show the information about your Tarantool application, write the external module `version` that outputs the information you need. The `tt version` call will execute this module instead of the built-in one:

```
tt version # Calls the external module if it's available
```

You can force the use of the internal module by running `tt` with the `--internal` or `-I` *option*. The following call will execute the built-in `version` even if there is an external module with the same name:

```
tt version -I # Calls the internal module
```

### tt interactive console

The `tt` utility features a command-line console that allows executing requests and Lua code interactively on the connected Tarantool instances. It is similar to the *Tarantool interactive console* with one key difference: the `tt` console allows connecting to any available instance, both local and remote. Additionally, it offers more flexible output formatting capabilities.

### Entering the console

To connect to a Tarantool instance using the `tt` console, run *tt connect*.

Specify the instance URI and the user credentials in the corresponding options:

```
$ tt connect 192.168.10.10:3301 -u myuser -p p4$$w0rD
   • Connecting to the instance...
   • Connected to 192.168.10.10:3301

192.168.10.10:3301>
```

If a user is not specified, the connection is established on behalf of the `guest` user.

If the instance runs in the same `tt` environment, you can establish a local connection with it by specifying the `<application>:<instance>` string instead of the URI:

```
$ tt connect app:storage001
    • Connecting to the instance...
    • Connected to app:storage001

 app:storage001>
```

Local connections are established on behalf of the `admin` user.

To get the list of supported console commands, enter `\help` or `?`. To quit the console, enter `\quit` or `\q`.

### Console input

Similarly to the *Tarantool interactive console*, the `tt` console can handle Lua or SQL input. The default is Lua. For Lua input, the tab-based autocompletion works automatically for loaded modules.

To change the input language to SQL, run `\set language sql`:

```
app:storage001> \set language sql
app:storage001> select * from bands where id = 1
---
- metadata:
  - name: id
    type: unsigned
  - name: band_name
    type: string
  - name: year
    type: unsigned
  rows:
  - [1, 'Roxette', 1986]
...
```

To change the input language back to Lua, run `\set language lua`:

```
app:storage001> \set language lua
app:storage001> box.space.bands:select { 1 }
---
- - [1, 'Roxette', 1986]
...
```

---

**Примечание:** You can also specify the input language in the `tt connect` call using the `-l`/`--language` option:

```
$ tt connect app:storage001 -l sql
```

---

### Console output

By default, the `tt` console prints the output data in the YAML format, each tuple on the new line:

```
app:storage001> box.space.bands:select { }
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'Ace of Base', 1987]
...
```

You can switch to alternative output formats – Lua or ASCII (pseudographics) tables – using the `\set output` console command:

```
app:storage001> \set output lua
app:storage001> box.space.bands:select { }
{{1, "Roxette", 1986}, {2, "Scorpions", 1965}, {3, "Ace of Base", 1987}};
app:storage001> \set output table
app:storage001> box.space.bands:select { }
+------+-------------+------+
| col1 | col2        | col3 |
+------+-------------+------+
| 1    | Roxette     | 1986 |
+------+-------------+------+
| 2    | Scorpions   | 1965 |
+------+-------------+------+
| 3    | Ace of Base | 1987 |
+------+-------------+------+
```

The table output can be printed in the transposed format, where an object's fields are arranged in columns instead of rows:

```
app:storage001> \set output ttable
app:storage001> box.space.bands:select { }
+------+---------+-----------+-------------+
| col1 | 1       | 2         | 3           |
+------+---------+-----------+-------------+
| col2 | Roxette | Scorpions | Ace of Base |
+------+---------+-----------+-------------+
| col3 | 1986    | 1965      | 1987        |
+------+---------+-----------+-------------+
```

---

**Примечание:** You can also specify the output format in the `tt connect` call using the `-x`/`--outputformat` option:

```
$ tt connect app:storage001 -x table
```

---

For `table` and `ttable` output, more customizations are possible with the following commands:

- `\set table_format` – table format: default (pseudographics, or ASCII table), Markdown, or Jira-compatible format:

```
app:storage001> \set table_format jira
app:storage001> box.space.bands:select {}
| col1 | 1 | 2 | 3 |
| col2 | Roxette | Scorpions | Ace of Base |
| col3 | 1986 | 1965 | 1987 |
```

- `\set grahpics` – enable or disable graphics for table cells in the default format:

```
app:storage001> \set table_format default
app:storage001> \set graphics false
app:storage001> box.space.bands:select {}
 col1  1        2          3
 col2  Roxette  Scorpions  Ace of Base
 col3  1986     1965       1987
```

- `\set table_column_width` – maximum column width.

```
app:storage001> \set table_column_width 6
app:storage001> box.space.bands:select {}
 col1  1       2       3
 col2  Roxett  Scorpi  Ace of
       +e      +ons    + Base
 col3  1986    1965    1987
```

### Commands

#### \help, ?

Show help on the `tt` console.

#### \quit, \q

Quit the `tt` console.

#### \shortcuts

Show available keyboard shortcuts.

#### \set language {lua|sql}

Set the input language. Possible values:

- `lua` (default)
- `sql`

An analog of the *tt connect* option `-l`/`--language`.

#### \set output FORMAT, \x{l|t|T|y}, \x

Set the output format. Possible `FORMAT` values:

- `yaml` (default) – each output item is a YAML object. Example: `[1, 'Roxette', 1986]`. Shorthand: `\xy`.
- `lua` – each output tuple is a separate Lua table. Example: `{{1, "Roxette", 1986}};`. Shorthand: `\xl`.
- `table` – the output is a table where tuples are rows. Shorthand: `\xt`.
- `ttable` – the output is a transposed table where tuples are columns. Shorthand: `\xT`.

---

**Примечание:** The `\x` command switches the output format cyclically in the order `yaml` > `lua` > `table` > `ttable`.

---

The format of `table` and `ttable` output can be adjusted using the `\set table_format`, `\set graphics`, and `\set table_colum_width` commands.

An analog of the *tt connect* option `-x`/`--outputformat`.

#### \set table_format

Set the table format if the output format is `table` or `ttable`. Possible values:

- `default` – a pseudographics (ASCII) table.
- `markdown` – a table in the Markdown format.
- `jira` – a Jira-compatible table.

#### \set graphics {true|false}, \x{g|G}

Whether to print pseudographics for table cells if the output format is `table` or `ttable`. Possible values: `true` (default) and `false`.

The shorthands are:

- `\xG` for `true`
- `\xg` for `false`

#### \set table_colum_width WIDTH, \xw WIDTH

Set the maximum printed width of a table cell content. If the length exceeds this value, it continues on the next line starting from the + (plus) sign.

Shorthand: `\xw`

#### tt environments

The key aspect of the `tt` usage is an *environment*. A `tt` environment is a directory that includes a `tt` configuration, Tarantool installations, application files, and other resources. If you're familiar with Python virtual environments, you can think of `tt` environments as their analog.

`tt` environments enable independent management of multiple Tarantool applications, each running on its own Tarantool version and configuration, on a single host in an isolated manner.

To create a `tt` environment in a directory, run *tt init* in it.

#### Multi-instance applications

`tt` supports Tarantool applications that run on multiple instances. For example, you can write an application that includes different source files for storage and router instances. With `tt`, you can start and stop them in a single call, or manage each instance independently.

Learn more about working with multi-instance applications in *Multi-instance applications*.

#### Replacement for tarantooctl and Cartridge CLI

A multi-purpose tool for working with Tarantool from the command line, `tt` has come to replace the deprecated utilities *tarantoolctl* and Cartridge CLI command-line utilities. The instructions on migration to `tt` are provided in the tt GitHub reposirory.

### 9.3.2 Tarantool Cluster Manager

---

**Enterprise Edition**

Tarantool Cluster Manager is a part of the Enterprise Edition.

---

Tarantool Cluster Manager (TCM) is a web-based visual tool for configuring, managing, and monitoring Tarantool EE clusters. It provides a GUI for working with clusters and individual instances, from monitoring their state to executing commands interactively in an instance's console.

TCM is a standalone application included in the Tarantool Enterprise Edition *distribution package*. It is shipped as ready-to-run executable for Linux platforms.

TCM works only with Tarantool EE clusters that use centralized configuration in *etcd* or a Tarantool-based configuration storage. When you create or edit a cluster's configuration in TCM, it publishes the saved configuration to the storage. This ensures consistent and reliable configuration storage. A single TCM installation can connect to multiple Tarantool EE clusters and switch between them in one click.

To provide enterprise-grade security, TCM features its own role-based access control. You can create users and assign them roles that include required permissions. For example, a user can be an administrator of a specific cluster or only have the right to read data. LDAP authorization is supported as well.

### Connecting clusters

**Enterprise Edition**

Tarantool Cluster Manager is a part of the Enterprise Edition.

Tarantool Cluster Manager works with clusters that:

- run on Tarantool EE 3.0 or later
- use *centralized configuration* stored in etcd or another Tarantool cluster.

A single TCM installation can have multiple connected clusters. A connection to TCM doesn't affect the cluster's functioning. You can connect clusters to TCM and disconnect them on the fly.

There are two ways to add a cluster to TCM:

- Connect an existing cluster.
- Add a new cluster and write its configuration from scratch in the TCM web UI.

In both cases, you need to deploy Tarantool and start the cluster instances using the *tt CLI utility* or another suitable way.

#### Connection parameters

When connecting a cluster to TCM, you need to provide two sets of connection parameters: for the cluster instances and for the centralized configuration storage.

#### Configuration storage connection

The cluster configuration can be stored in either an *etcd* cluster or a separate Tarantool-based storage. In both cases, the following connection parameters are required:

- A key prefix used to identify the cluster in the configuration storage. A prefix must be unique for each cluster in storage.
- URIs of all instances of the configuration storage.
- The credentials for accessing the configuration storage: an etcd user or a *Tarantool user*.

Additionally, if SSL or TLS encryption is enabled for the configuration storage, provide the corresponding encryption configuration: keys, certificates, and other parameters. For the complete list of parameters, consult the etcd documentation or Tarantool *Securing connections with SSL*.

#### Cluster connection

For interaction with the cluster instances, TCM needs the following access parameters:

- A *Tarantool user* that exists in the cluster and their password. TCM connects to the cluster on behalf of this user.
- An *SSL configuration* if the traffic encryption is enabled on the cluster.

### Managing connected clusters

Administrators can add new clusters, edit, and remove existing ones from TCM.

Connected clusters are listed on the **Clusters** page.

### Connecting a pre-configured cluster

If you already have a cluster and want to connect it to TCM, follow these steps:

1. Go to **Clusters** and click **Add**.
2. Fill in the general cluster information:
   - Specify an arbitrary name.
   - Optionally, provide a description and select a color to mark this cluster in TCM.
   - Optionally, enter the URLs of additional services for the cluster. For example, a Grafana dashboard that monitors the cluster metrics, or a syslog server for viewing the cluster logs. TCM provides quick access to these URLs on the cluster **Stateboard** page.
3. Provide the details of the cluster configuration storage:
   - Storage type: **etcd** or **tarantool**.
   - The **Prefix** specified in the cluster configuration.
   - The URIs of the configuration storage instances.
   - The credentials for accessing the configuration storage.
   - The SSL/TLS parameters if the connection encryption is enabled on the storage.
4. Provide the credentials for accessing the cluster: a Tarantool user's name, their password, and SSL parameters in case *traffic encryption* is enabled on the cluster.

### Adding a new cluster

If you don't have a cluster yet, you can add one in TCM and write its configuration from scratch using the *built-in configuration editor*.

---

**Важно:** When adding a new cluster, you need to have a storage for its configuration up and running so that TCM can connect to it. Cluster instances can be deployed later.

---

To add a new cluster:

1. Go to **Clusters** and click **Add**.
2. Fill in the general cluster information:
   - Specify an arbitrary name.
   - Optionally, provide a description and select a color to mark this cluster in TCM.
   - Optionally, enter the URLs of additional services for the cluster. For example, a Grafana dashboard that monitors the cluster metrics, or a syslog server for viewing the cluster logs. TCM provides quick access to these URLs on the cluster **Stateboard** page.
3. Select the type of the cluster configuration storage: **etcd** or **tarantool**.

4. Define a unique **Prefix** for identifying this cluster in the configuration storage.
5. Provide the connection details for the cluster configuration storage:
   - The URIs of configuration storage instances.
   - The credentials for accessing the configuration storage.
   - The SSL/TLS parameters if the connection encryption is enabled on the storage.
6. Provide the cluster credentials: a username, a password, and SSL parameters in case *traffic encryption* is enabled on the cluster.

Once you add the cluster:

- Set up the cluster configuration using the TCM *configuration editor*.
- Deploy Tarantool on the cluster nodes using the *tt CLI utility* or other suitable tools.
- Start the cluster using the *tt CLI utility* or other suitable tools.

### Editing a connected cluster

To edit a connected cluster, go to **Clusters** and click **Edit** in the **Actions** menu of the corresponding table row.

### Disconnecting a cluster

To disconnect a cluster from TCM, go to **Clusters** and click **Disconnect** in the **Actions** menu of the corresponding table row.

---

**Примечание:** Disconnecting a cluster does not affect its functioning. The only thing that changes is that it's no longer shown in TCM. You can connect this cluster again at any time.

---

### Configuring clusters

---

**Enterprise Edition**

Tarantool Cluster Manager is a part of the Enterprise Edition.

---

Tarantool Cluster Manager features a built-in text editor for Tarantool EE cluster configurations.

When you connect a cluster to TCM, it gains access to the cluster's centralized configuration storage: an etcd or a Tarantool cluster. TCM has both read and write access to the cluster configuration. This enables the configuration editor to work in two ways:

- If a configuration already exists, the editor shows its current state.
- When you change the configuration in the editor and apply changes, they are sent to the configuration storage.

To learn how to write Tarantool cluster configurations, see *Configuration*.

### Managing a cluster's configuration

The configuration editor is available on the **Cluster** > **Configuration** page.

To start managing a cluster's configuration, select this cluster in the **Cluster** drop-down and go to the **Configuration** page.

A cluster configuration in TCM can consist of one or multiple YAML files. When there are multiple files, they are all considered parts of a single cluster configuration. You can use this for structuring big cluster configurations. All files that form the configuration of a cluster are listed on the left side of the **Cluster configuration** page.

To add a cluster configuration file, click the plus icon (+) below the page title.

To open a configuration file in the editor, click its name in the file list.

To delete a cluster configuration file, click the **Delete** button beside the filename.

To download a cluster configuration file, click the **Download** button beside the filename.

**Предупреждение:** All configuration changes are discarded when you leave the **Cluster configuration** page. *Save* the configuration if you want to continue editing it later or *apply* it to start using it on the cluster.

### Saving a configuration draft

TCM can store configurations drafts. If you want to leave an unfinished configuration and return to it later, save it in TCM. Saving applies to whole cluster configurations: it records the edits of all files, file additions, and file deletions.

To save a cluster configuration draft after editing, click **Save** in the **Cluster configuration** page.

All unsaved changes are discarded when you leave the **Cluster configuration** page.

If you have a saved configuration draft, you can reset the changes for each of its files individually. A reset returns the file into the state that is currently used by a cluster (that is, saved in the configuration storage). If you reset a newly added file, it is deleted.

To reset a saved configuration file, click the **Reset** button beside the filename.

### Applying a configuration

When you finish editing a configuration and it's ready to use, apply the updated configuration to the cluster. To apply a cluster configuration, click **Apply** on the **Cluster configuration** page. This sends the new configuration to the cluster configuration storage, and it comes into effect upon the cluster configuration reload.

### Cluster monitoring

Tarantool Cluster Manager provides means for monitoring various aspects of connected clusters, such as:

- topology
- instance state
- memory usage
- data distribution
- Tarantool versions

Cluster monitoring tools are available on the **Cluster** > **Stateboard** page.

### Cluster topology

The cluster topology is displayed on the **Stateboard** page in one of two forms: a list or a graph.

#### List view

The list view of the cluster topology is used by default. In this view, each row contains the general information about an instance: its current state, memory usage and limit, and other parameters.

In the list view, TCM additionally displays the Tarantool version information and instance states on circle diagrams. You can click the sectors of these diagrams to filter the instances with the selected versions and states.

To switch to the list view, click the list button on the right of the search bar on the **Stateboard** page.

#### Graph view

The graph view of the cluster topology is shown in a tree-like structure where leafs are the cluster's instances. Each instance's state is shown by its color. You can move the graph vertexes to arrange them as you like, and zoom in and out, which is helpful for bigger clusters.

To switch to the graph view, click the graph button on the right of the search bar on the **Stateboard** page.

### Instance grouping

By default, the cluster topology is shown hierarchically as it's defined in the configuration: instances are grouped by their replica set, and replica sets are grouped by their configuration group.

For better navigation across the cluster, you can adjust the instance grouping. For example, you can group instances by their roles or custom tags defined in the configuration. A typical case for such tags is adding a geographical markers to instances. In this case, you see if issues happen in a specific data center or server.

To change the instance grouping, click **Group by** in the **Actions** menu on the **Stateboard** page. Then add or remove grouping criteria.

### Filtering

You can filter the instances shown on the **Stateboard** page using the search bar at the top. It has predefined filters that select:

- instances with errors or warnings
- leader or read-only instances
- instances with no issues
- stale instances

To display all instances, delete the filter applied in the search bar.

### Instance details

The general information about the state of cluster instances is shown in the list view of the cluster topology. Each row contains the information about the instance status, used and available memory, read-only status, and virtual buckets for sharded clusters.

To view the detailed information about an instance or connect to it, click the corresponding row in the instances list or a vertex of the graph. On the instance page, you can find:

- its configuration overview
- current state (with warning and error messages if any)
- the detailed Tarantool information returned by the instance introspection functions from *box.info*, *box.stat*, and other built-in modules.

Additionally, on the instance details page there is a terminal in which you can execute arbitrary Lua code on the instance.

### Additional monitoring services

When you *connect a cluster* to TCM, you can specify URLs of external services linked to this cluster. For example, this can be a Grafana server that monitors the cluster metrics.

All the URLs added for a cluster are available for quick access in the **Actions** menu on the **Stateboard** page.

### Access control

---

**Enterprise Edition**

Tarantool Cluster Manager is a part of the Enterprise Edition.

---

Tarantool Cluster Manager features a role-based access control system. It enables flexible management of access to TCM functions, connected clusters, and stored data. The TCM access system uses three main entities: permissions, roles, and users (or user accounts). They work as follows:

- Permissions correspond to specific functions or objects in TCM (*administrative permissions*) or operations on clusters (*cluster permissions*).
- Roles are predefined sets of *administrative* permissions to assign to users.

- Users have roles that define their access rights to TCM functions and objects, and *cluster* permissions that are assigned for each cluster separately.

---

**Примечание:** TCM users, roles, and permissions are not to be confused with similar subjects of the *Tarantool access control system*. To access Tarantool instances directly, Tarantool users with corresponding roles are required.

---

### Permissions

Permissions define access to specific actions that users can do in TCM. For example, there are permissions to view connected clusters or to manage users.

There are two types of permissions in TCM: *administrative* and *cluster* permissions.

- *Administrative permissions* provide access to TCM functions. They define which pages and controls are available to users in the web UI. Typically, *read* permissions define pages shown in the left menu. *Write* permissions define the availability of controls for managing objects on the pages. For example, users with read permission to clusters can view the **Clusters** page but they don't see **Add**, **Edit**, or **Remove** buttons unless they have the write permission.

  Administrative permissions are assigned to users through *roles*.

- *Cluster permissions* enable actions with connected Tarantool clusters. These permissions are granted to users on a per-cluster level: each user has a separate set of permissions for each cluster.

  Cluster permissions define which pages of the **Cluster** menu section users see and what actions they can take on these pages. For example, users with the read configuration permission to a cluster configuration see the **Configuration** page when this cluster is selected.

  Cluster permissions are assigned to users individually when creating or editing them.

Permissions are predefined in TCM, there is no way to change, add, or delete them. The complete lists of administrative and cluster permissions in TCM are provided in the *Permissions reference*.

### Roles

Roles are groups of *administrative permissions* that are assigned to users together.

The assigned roles define pages that users see in TCM and actions available on these pages.

---

**Примечание:** Roles don't include cluster permissions. Access to connected clusters is configured for each user individually.

---

### Default roles

TCM comes with default roles that cover three common usage scenarios:

- **Super Admin Role** is a default role with all available *administrative permissions*. Additionally, the users with this role automatically gain all *cluster permissions* to all clusters.
- **Cluster Admin Role** is a default role for cluster administration. It includes administrative permissions for cluster management.
- **Default User Role** is a default role for working with clusters. It includes basic administrative *read* permissions that are required to log in to TCM and navigate to a cluster.

### Managing roles

Administrators can create new roles, edit, and delete existing ones.

Roles are listed on the **Roles** page.

To create a new role, click **Add**, enter the role name, and select the permissions to include in the role.

To edit an existing role, click **Edit** in the **Actions** menu of the corresponding table row.

To delete a role, click **Delete** in the **Actions** menu of the corresponding table row.

---

**Примечание:** You can delete a role only if there are no users with this role.

---

### Users

TCM users gain access to objects and actions through assigned *roles* and *cluster permissions*.

A user can have any number of roles or none of them. Users without roles have access only to clusters that are assigned to them.

TCM uses password authentication for users. For information on password management, see the *Passwords* section below.

### Default admin

There is one default user **Default Admin**. It has all the available permissions, both administrative and cluster ones. When new clusters are added in TCM, **Default Admin** automatically receives all cluster permissions for them as well.

### Managing users

Administrators can create new users, edit, and delete existing ones.

The tools for managing users are located on the **Users** page.

To create a user:

1. Click **Add**.
2. Fill in the user information: username, full name, and description.

3. Generate or type in a password.
4. Select roles to assign to the user.
5. Add clusters to give the user access to, and select cluster permissions for each of them.

To edit a user, click **Edit** in the **Actions** menu of the corresponding table row.

To delete a user, click **Delete** in the **Actions** menu of the corresponding table row.

### Passwords

TCM uses the general term *secret* for user authentication keys. A secret is any pair of a public and a private key that can be used for authentication. In TCM 1.0.0, *password* is the only supported secret type. In this case, the public key is a username, and the private key is a password.

Users receive their first passwords during their *account creation*.

All passwords are governed by the *password policy*. It can be flexibly configured to follow the security requirements of your organization.

### Changing your password

To change your own password, click your name in the top-right corner and go to **Settings** > **Change password**.

### Changing users“ passwords

Administrators can manage a user's password on this user's **Secrets** page. To open it, click **Secrets** in the **Actions** menu of the corresponding **Users** table row.

To change a user's password, click **Edit** in the **Actions** menu of the corresponding **Secrets** table row and enter the new password in the **New secret key** field.

### Password expiry

Passwords expire automatically after the expiration period defined in the *password policy*. When a user logs in to TCM with an expired password, the only action available to them is a password change. All other TCM functions and objects are unavailable until the new password is set.

Administrators can also set users“ passwords to expired manually. To set a user's password to expired, click **Expire** in the **Actions** menu of the corresponding **Secrets** table row.

---

**Важно:** Password expiration can't be reverted.

---

### Blocking passwords

To forbid users“ access to TCM, administrators can temporarily block their passwords. A blocked password can't be used to log into TCM until it's unblocked manually or the blocking period expires.

To block a user's password, click **Block** in the **Actions** menu of the corresponding **Secrets** table row. Then provide a blocking reason and enter the blocking period.

To unblock a blocked password, click **Unblock** in the **Actions** menu of the corresponding **Secrets** table row.

### Password policy

Password policy helps improve security and comply with security requirements that can apply to your organization.

You can edit the TCM password policy on the **Password policy** page. There are the following password policy settings:

- **Minimal password length**.
- **Do not use last N passwords**.
- **Password expiration in days**. Users“ passwords *expire* after this number of days since they were set. Users with expired passwords lose access to any objects and functions except password change until they set a new password.
- **Password expiration warning in days**. After this number of days, the user sees a warning that their password expires soon.
- **Block after N login attempts**. Temporarily block users if they enter their username or password incorrectly this number of times consecutively.
- **User lockout time in seconds**. The time interval for which users can't log in after spending all failed login attempts.
- **Password must include**. Characters and symbols that must be present in passwords:
  - **Lowercase characters (a-z)**
  - **Uppercase characters (A-Z)**
  - **Digits (0-9)**
  - **Symbols (such as !@#$%^&*()_+№»“:,.;=][{}`?>/.)**

### Sessions

Administrators can view and revoke user sessions in TCM. All active sessions are listed on the **Sessions** page. To revoke a session, click **Revoke** in the **Actions** menu of the corresponding table row.

To revoke all sessions of a user, go to **Users** and click **Revoke all sessions** in the **Actions** menu of the corresponding table row.

### Permissions reference

#### Administrative permissions

The following administrative permissions are available in TCM:

| Permission | Description |
|---|---|
| `admin.clusters.read` | View connected clusters“ details |
| `admin.clusters.write` | Edit cluster details and add new clusters |
| `admin.users.read` | View users“ details |
| `admin.users.write` | Edit user details and add new users |
| `admin.roles.read` | View roles“ details |
| `admin.roles.write` | Edit roles and add new roles |
| `admin.addons.read` | View add-ons |
| `admin.addons.write` | Edit add-on flags |
| `admin.addons.upload` | Upload new add-ons |
| `admin.auditlog.read` | View audit log configuration and read audit log in TCM |
| `admin.auditlog.write` | Edit audit log configuration |
| `admin.sessions.read` | View users“ sessions |
| `admin.sessions.write` | Revoke users“ sessions |
| `admin.ldap.read` | View LDAP configurations |
| `admin.ldap.write` | Manage LDAP configurations |
| `admin.passwordpolicy.read` | View password policy |
| `admin.passwordpolicy.write` | Manage password policy |
| `admin.devmode.toggle` | Toggle development mode |
| `admin.secrets.read` | View information about users“ secrets |
| `admin.secrets.write` | Manage users“ secrets: add, edit, expire, block, delete |
| `user.password.change` | User's permission to change their own password |
| `admin.lowlevel.state.read` | Read low-level information from TCM storage (for debug purposes) |
| `admin.lowlevel.state.write` | Write low-level information to TCM storage (for debug purposes) |

#### Cluster permissions

The following cluster permissions are available in TCM:

| Permission | Description |
|---|---|
| `cluster.config.read` | View cluster configuration |
| `cluster.config.write` | Manage cluster configuration |
| `cluster.stateboard.read` | View cluster stateboard |
| `cluster.explorer.read` | Read data from cluster instances |
| `cluster.explorer.write` | Write data to cluster instances |
| `cluster.explorer.call` | Execute stored functions on cluster instances |
| `cluster.explorer.eval` | Execute code on cluster instances |
| `cluster.space.read` | Read cluster data schema |
| `cluster.space.write` | Modify cluster data schema |
| `cluster.lowlevel.state.read` | Read low-level information about cluster configuration (for debug purposes) |
| `cluster.lowlevel.state.write` | Write low-level information about cluster configuration (for debug purposes) |

### Audit log

---

**Enterprise Edition**

Tarantool Cluster Manager is a part of the Enterprise Edition.

---

Tarantool Cluster Manager provides the audit logging functionality for tracking user activity and security-related events, such as:

- Successful and failed login attempts.
- Access to clusters, their configurations, data models, and stored data.
- Changes in the access control system: users, roles, passwords, LDAP configurations.

The complete list of TCM audit events is provided in *Event types*.

---

**Примечание:** TCM audit log records only events that happen in TCM itself. For information about Tarantool audit logging, see *Audit module*.

---

Audit logging is disabled in TCM by default. To start recording events, you need to *enable* and *configure* it.

The audit log stores event details in the JSON format. Each log entry contains the event type, description, time, impacted objects, and other information that may be used for incident investigation. The complete list of fields is provided in *Structure of audit log events*.

TCM also provides a built-in interface for reading and searching the audit log. For details, see *Viewing audit log*.

#### Enabling audit logging

To enable audit logging in TCM, go to **Audit settings** and click **Enable**.

To additionally send audit log events to the standard output, click **Send to stdout**.

#### Audit log configuration

TCM audit events can be logged to a local file or sent to a syslog server. To configure audit logging, go to **Audit settings**.

#### Writing to a file

To write TCM audit logs to a file:

1. Go to **Audit settings** and select the **file** protocol.
2. Specify the name of the audit log file. The file appears in the TCM working directory.
3. Configure the log files rotation: the maximum file size and age, and the number of files to store simultaneously.
4. (Optional) Enable compression of audit log files.

Configuration parameters:

- **Output file name**. The name of the audit log file. Default: `audit.log`
- **Max size (in MB)**. The maximum size of the log file before it gets rotated, in megabytes. Default: 100.
- **Max backups**. The maximum number of stored audit log files. Default: 10.
- **Max age (in days)**. The maximum age of audit log files in days. Default: 30.
- **Compress**. Compress audit log files into `gzip` archives when rotating.

### Sending to syslog

If you use a centralized log management system based on syslog, you can configure TCM to send its audit log to your syslog server:

1. Go to **Audit settings** and select the **syslog** protocol.
2. Enter the syslog server URI and select the network protocol. Typically, `syslogd` listens on port 514 and uses the UDP protocol.
3. Specify the syslog logging parameters: timeout, priority, and facility.

Configuration parameters:

- **Protocol**. The network protocol used for connecting to the syslog server. Default: `udp`.
- **Output**. The syslog server URI. Default: 127.0.0.1:514 (localhost).
- **Timeout**. The syslog write timeout in the ISO 8601 duration format. Default: `PT2S` (two seconds).
- **Priority**. The syslog severity level. Default: `info`.
- **Facility**. The syslog facility. Default: `local0`.

### Selecting audit events to record

When the audit log is enabled, TCM records all audit events listed in *Event types*. To decrease load and make the audit log comply with specific security requirements, you can record only selected events. For example, these can be events of user account management or events of cluster data access.

To select events to record into the audit log, go to **Audit settings** and enter their *types* into the **Filters** field one-by-one, pressing the **Enter** key after each type.

To remove an event type from a filters list, click the cross icon beside it.

### Viewing audit log

If the audit log is written to a file, you can view it in TCM on the **Audit log** page. On this page, you can view or search for events.

To view the details of a logged audit event, click the corresponding line in the table.

To search for an event, use the search bar at the top of the page. Note that the search is case-sensitive. For example, to find events with the `ALARM` severity, enter `ALARM`, not `alarm`.

### Structure of audit log events

All entries of the TCM audit log include the mandatory fields listed in the table below.

| Field | Description | Example |
|---|---|---|
| `time` | Time of the event | 2023-11-23T12:05:27.099+07:00 |
| `severity` | Event severity: `VERBOSE`, `INFO`, `WARNING`, or `ALARM` | INFO |
| `type` | Audit *event type* | user.update |
| `description` | Human-readable event description | Update user |
| `uuid` | Event UUID | f8744f51-5760-40c3-ae2d-0b4d6b44836f |
| `user` | UUID of the user who triggered the event | 942a4f54-cf7f-4f46-80ce-3511dbbb57b7 |
| `remote` | Remote host that triggered the event | 100.96.163.226:48722 |
| `host` | The TCM host on which the event happened | 100.96.163.226:8080 |
| `userAgent` | Information about the client application and platform that was used to trigger the event | Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36 |
| `permission` | The permission that was used to trigger the event | [«admin.users.write»] |
| `result` | Event result: `ok` or `nok` | ok |
| `err` | Human-readable error description for events with `nok` result | failed to login |
| `fields` | Additional fields for specific event types in the key-value format | Key examples:<br>• `clusterId` in cluster-related events<br>• `payload` in events that include sending data to the server<br>• `username` in `current.*` or `auth.*` events |

This is an example of an audit log entry on a successful login attempt:

```
{
    "time": "2023-11-23T12:01:27.247+07:00",
    "severity": "INFO",
    "description": "Login user",
    "type": "current.login",
    "uuid": "4b9c2dd1-d9a1-4b40-a448-6bef4a0e5c79",
    "user": "",
    "remote": "127.0.0.1:63370",
    "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36␣
↪(KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36",
    "host": "127.0.0.1:8080",
    "permissions": [],
    "result": "ok",
    "fields": [
        {
            "Key": "username",
            "Value": "admin"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        },
        {
            "Key": "method",
            "Value": "null"
        },
        {
            "Key": "output",
            "Value": "true"
        }
    ]
}
```

### Event types

The following table lists all possible values of the `type` field of TCM audit log events.

| Event type | Description |
|---|---|
| `auth.fail` | Authentication failed |
| `auth.ok` | Authentication successful |
| `access.denied` | An attempt to access an object without the required permission |
| `user.add` | User added |
| `user.update` | User updated |
| `user.delete` | User deleted |
| `secret.add` | User secret added |
| `secret.update` | User secret updated |
| `secret.block` | User secret blocked |
| `secret.unblock` | User secret unblocked |
| `secret.delete` | User secret deleted |
| `secret.expire` | User secret expired |
| `session.revoke` | Session revoked |
| `session.revokeuser` | All user's sessions revoked |
| `explorer.insert` | Data inserted in a cluster |
| `explorer.delete` | Data deleted from a cluster |
| `explorer.replace` | Data replaced in a cluster |
| `explorer.call` | Stored function called on a cluster |
| `explorer.evaluate` | Code executed on a cluster |
| `explorer.switchover` | Master switched manually |
| `test.devmode` | Switched to development mode |
| `auditlog.config` | Audit log configuration changed |
| `passwordpolicy.save` | Password policy changed |
| `passwordpolicy.resetpasswords` | All passwords expired by an administrator |
| `ddl.save` | Cluster data model saved |
| `ddl.apply` | Cluster data model applied |
| `cluster.config.save` | Cluster configuration saved |
| `cluster.config.reset` | Saved cluster configuration reset |
| `cluster.config.apply` | Cluster configuration applied |
| `current.logout` | User logged out their own session |
| `current.revoke` | User revoked their own session |
| `current.revokeall` | User revoked all their active sessions |

continues on next page

Таблица 2 – продолжение с предыдущей страницы

| Event type | Description |
|---|---|
| `current.changepassword` | User changed their password |
| `role.add` | Role added |
| `role.update` | Role updated |
| `role.delete` | Role deleted |
| `cluster.add` | Cluster added |
| `cluster.update` | Cluster updated |
| `cluster.delete` | Cluster removed |
| `ldap.testlogin` | Login test executed for a LDAP configuration |
| `ldap.testconnection` | Connection test executed for a LDAP configuration |
| `ldap.add` | LDAP configuration added |
| `ldap.update` | LDAP configuration updated |
| `ldap.delete` | LDAP configuration deleted |
| `addon.enable` | Add-on enabled |
| `addon.disable` | Add-on disabled |
| `addon.delete` | Add-on removed |
| `tcmstate.save` | Low-level information saved in the TCM storage (for debug purposes) |
| `tcmstate.delete` | Low-level information deleted from the TCM storage (for debug purposes) |

### Configuration

**Enterprise Edition**

Tarantool Cluster Manager is a part of the Enterprise Edition.

This topic describes how to configure Tarantool Cluster Manager. For the complete list of TCM configuration parameters, see the *TCM configuration reference*.

#### Configuration structure

Tarantool Cluster Manager configuration is a set of parameters that define various aspects of TCM functioning. Parameters are grouped by the particular aspect that they affect. There are the following groups:

- HTTP
- logging
- configuration storage
- security
- add-ons
- limits
- TCM running mode

Parameter groups can be nested. For example, in the `http` group there are `tls` and `websession-cookie` groups, which define TLS encryption and cookie settings.

Parameter names are the full paths from the top-level group to the specific parameter. For example:

- `http.host` is the `host` parameter that is defined directly in the `http` group.

- `http.tls.enabled` is the `enabled` parameter that is defined in the `tls` nested group within `http`.

### Ways to pass configuration parameters

There are three ways to pass TCM configuration parameters:

- a YAML file
- environment variables
- command-line options of the TCM executable

### YAML file

TCM configuration can be stored in a YAML file. Its structure must reflect the configuration parameters hierarchy.

The example below shows a fragment of a TCM configuration file:

```
# a fragment of a YAML configuration file
cluster: # top-level group
    on-air-limit: 4096
    connection-rate-limit: 512
    tarantool-timeout: 10s
    tarantool-ping-timeout: 5s
http: # top-level group
    basic-auth: # nested group
        enabled: false
    network: tcp
    host: 127.0.0.1
    port: 8080
    request-size: 1572864
    websocket: # nested group
        read-buffer-size: 16384
        write-buffer-size: 16384
        keepalive-ping-interval: 20s
        handshake-timeout: 10s
        init-timeout: 15s
```

To start TCM with a YAML configuration, pass the location of the configuration file in the `-c` command-line option:

```
tcm -c=config.yml
```

### Environment variables

TCM can take values of its configuration parameters from environment variables. The variable names start with `TCM_`. Then goes the full path to the parameter, converted to upper case. All delimiters are replaced with underscores (_). Examples:

- `TCM_HTTP_HOST` is a variable for the `http.host` parameter.
- `TCM_HTTP_WEBSESSION_COOKIE_NAME` is a variable for the `http.websession-cookie.name` parameter.

The example below shows how to start TCM with configuration parameters passed in environment variables:

```
export TCM_HTTP_HOST=0.0.0.0
export TCM_HTTP_PORT=8888
tcm
```

### Command-line arguments

The TCM executable has `--` command-line options for each configuration parameter. Their names reflect the full path to the parameter, with all delimiters replaced by hyphens (-). Examples:

- `--http-host` is an option for `http.host`.
- `--http-websession-cookie-name` is an option for `http.websession-cookie.name`.

The example below shows how to start TCM with configuration parameters passed in command-line options:

```
./tcm --storage.etcd.embed.enabled --addon.enabled --http.host=0.0.0.0 --http.port=8888
```

### Configuration precedence

TCM configuration options are applied from multiple sources with the following precedence, from highest to lowest:

1. `tcm` executable arguments.
2. `TCM_*` environment variables.
3. Configuration from a YAML file.

If the same option is defined in two or more locations, the option with the highest precedence is applied. For options that aren't defined in any location, the default values are used.

You can combine different ways of TCM configuration for efficient management of multiple TCM installations:

- A single YAML file for all installations can contain the common configuration parts. For example, a single configuration storage that is used for all installations, or TLS settings.
- Environment variables that set specific parameters for each server, such as local directories and paths.
- Command-line options for parameters that must be unique for different TCM instances running on a single server. For example, `http.port`.

### Configuration parameter types

TCM configuration parameters have the Go language types. Note that this is different from the *Tarantool configuration parameters*, which have Lua types.

Most options have the Go's basic types: `int` and other numeric types, `bool`, `string`.

```
http:
    basic-auth:
        enabled: false # bool
    network: tcp # string
    host: 127.0.0.1 # string
    port: 8080 # int
    request-size: 1572864 # int64
```

Parameters that can take multiple values are arrays. In YAML, they are passed as YAML arrays: each item on a new line, starting with a dash.

```
storage:
provider: etcd
etcd:
    endpoints: # array
        - https://192.168.0.1:2379 # item 1
        - https://192.168.0.2:2379 # item 2
```

---

**Примечание:** In environment variables and command line options, such arrays are passed as semicolon-separated strings of items.

---

Parameters that set timeouts, TTLs, and other duration values, have the Go's time.Duration type. Their values can be passed in time-formatted strings such as `4h30m25s`.

```
cluster:
    tarantool-timeout: 10s # duration
    tarantool-ping-timeout: 5s # duration
```

Finally, there are parameters whose values are constants defined in Go packages. For example, *http.websession-cookie.same-site* values are constants from the Go's http.SameSite type. To find out the exact values available for such parameters, refer to the Go packages documentation.

```
http:
    websession-cookie:
        same-site: SameSiteStrictMode
```

### Creating a configuration template

You can create a YAML configuration template for TCM with all parameters and their default values using the `generate-config` option of the `tcm` executable.

To write a default TCM configuration to the `tcm.example.yml` file, run:

```
tcm generate-config > tcm.example.yml.
```

### Configuration reference

---

**Enterprise Edition**

Tarantool Cluster Manager is a part of the Enterprise Edition.

---

This topic describes *configuration parameters* of Tarantool Cluster Manager.

There are the following groups of TCM configuration parameters:

- *cluster*
- *http*
- *log*
- *storage*
- *addon*
- *limits*
- *security*
- *mode*

#### cluster

The `cluster` group defines parameters of TCM interaction with connected Tarantool clusters.

- *on-air-limit*
- *connection-rate-limit*
- *tarantool-timeout*
- *tarantool-ping-timeout*

`cluster.on-air-limit`
: The maximum number of on-air requests from TCM to all connected clusters.

  Type: int64
  Default: 4096
  Environment variable: TCM_CLUSTER_ON_AIR_LIMIT
  Command-line option: `--cluster-on-air-limit`

`cluster.connection-rate-limit`
: A rate limit for connections to Tarantool instances.

  Type: uint
  Default: 512
  Environment variable: TCM_CLUSTER_CONNECTION_RATE_LIMIT
  Command-line option: `--cluster-connection-rate-limit`

`cluster.tarantool-timeout`
: A timeout for receiving a response from Tarantool instances.

  Type: time.Duration
  Default: 10s
  Environment variable: TCM_CLUSTER_TARANTOOL_TIMEOUT
  Command-line option: `--cluster-tarantool-timeout`

`cluster.tarantool-ping-timeout`
: A timeout for receiving a ping response from Tarantool instances.

  Type: time.Duration
  Default: 5s
  Environment variable: TCM_CLUSTER_TARANTOOL_PING_TIMEOUT
  Command-line option: `--cluster-tarantool-ping-timeout`

### http

The `http` group defines parameters of HTTP connections between TCM and clients.

- *http.basic_auth.enabled*
- *http.network*
- *http.host*
- *http.port*
- *http.request-size*
- *http.websocket.read-buffer-size*
- *http.websocket.write-buffer-size*
- *http.websocket.keepalive-ping-interval*
- *http.websocket.handshake-timeout*
- *http.websocket.init-timeout*

- *http.websession-cookie.name*
- *http.websession-cookie.path*
- *http.websession-cookie.domain*
- *http.websession-cookie.ttl*
- *http.websession-cookie.secure*
- *http.websession-cookie.http-only*
- *http.websession-cookie.same-site*
- *http.cors.enabled*
- *http.cors.allowed-origins*
- *http.cors.allowed-methods*
- *http.cors.allowed-headers*
- *http.cors.exposed-headers*
- *http.cors.allow-credentials*
- *http.cors.debug*
- *http.metrics-endpoint*
- *http.tls.enabled*
- *http.tls.cert-file*
- *http.tls.key-file*
- *http.tls.server*
- *http.tls.min-version*
- *http.tls.max-version*
- *http.tls.curve-preferences*
- *http.tls.cipher-suites*
- *http.read-timeout*
- *http.read-header-timeout*
- *http.write-timeout*
- *http.idle-timeout*
- *http.idle-timeout*
- *http.disable-general-options-handler*
- *http.max-header-bytes*
- *http.api-timeout*
- *http.api-update-interval*
- *http.frontend-dir*
- *http.show-stack-trace*
- *http.trace*
- *http.max-static-size*

- *http.graphql.complexity*

`http.basic_auth.enabled`
Whether to use the HTTP basic authentication.

Type: bool
Default: false
Environment variable: TCM_HTTP_BASIC_AUTH_ENABLED
Command-line option: `--http-basic-auth-enabled`

`http.network`
An addressing scheme that TCM uses.

Possible values:

- `tcp`: IPv4 address
- `tcp6`: IPv6 address
- `unix`: Unix domain socket

Type: string
Default: tcp
Environment variable: TCM_HTTP_NETWORK
Command-line option: `--http-network`

`http.host`
A host name on which TCM serves.

Type: string
Default: 127.0.0.1
Environment variable: TCM_HTTP_HOST
Command-line option: `--http-host`

`http.port`
A port on which TCM serves.

Type: int
Default: 8080
Environment variable: TCM_HTTP_PORT
Command-line option: `--http-port`

`http.request-size`
: The maximum size (in bytes) of a client HTTP request to TCM.

  Type: int64
  Default: 1572864
  Environment variable: TCM_HTTP_REQUEST_SIZE
  Command-line option: `--http-request-size`

`http.websocket.read-buffer-size`
: The size (in bytes) of the read buffer for WebSocket connections.

  Type: int
  Default: 16384
  Environment variable: TCM_HTTP_WEBSOCKET_READ_BUFFER_SIZE
  Command-line option: `--http-websocket-read-buffer-size`

`http.websocket.write-buffer-size`
: The size (in bytes) of the write buffer for WebSocket connections.

  Type: int
  Default: 16384
  Environment variable: TCM_HTTP_WEBSOCKET_WRITE_BUFFER_SIZE
  Command-line option: `--http-websocket-write-buffer-size`

`http.websocket.keepalive-ping-interval`
: The time interval for sending WebSocket keepalive pings.

  Type: time.Duration
  Default: 20s
  Environment variable: TCM_HTTP_WEBSOCKET_KEEPALIVE_PING_INTERVAL
  Command-line option: `--http-websocket-keepalive-ping-interval`

`http.websocket.handshake-timeout`
: The time limit for completing a WebSocket opening handshake with a client.

  Type: time.Duration
  Default: 10s

Environment variable: TCM_HTTP_WEBSOCKET_HANDSHAKE_TIMEOUT
Command-line option: `--http-websocket-handshake-timeout`

`http.websocket.init-timeout`
The time limit for establishing a WebSocket connection with a client.

Type: time.Duration
Default: 15s
Environment variable: TCM_HTTP_WEBSOCKET_INIT_TIMEOUT
Command-line option: `--http-websocket-init-timeout`

`http.websession-cookie.name`
The name of the cookie that TCM sends to clients.

This value is used as the cookie name in the Set-Cookie HTTP response header.

Type: string
Default: tcm
Environment variable: TCM_HTTP_WEBSESSION_COOKIE_NAME
Command-line option: `--http-websession-cookie-name`

`http.websession-cookie.path`
The URL path that must be present in the requested URL in order to send the cookie.

This value is used in the `Path` attribute of the Set-Cookie HTTP response header.

Type: string
Default: «»
Environment variable: TCM_HTTP_WEBSESSION_COOKIE_PATH
Command-line option: `--http-websession-cookie-path`

`http.websession-cookie.domain`
The domain to which the cookie can be sent.

This value is used in the `Domain` attribute of the Set-Cookie HTTP response header.

Type: string
Default: «»
Environment variable: TCM_HTTP_WEBSESSION_COOKIE_DOMAIN
Command-line option: `--http-websession-cookie-domain`

`http.websession-cookie.ttl`
: The maximum lifetime of the TCM cookie.

  This value is used in the `Max-Age` attribute of the Set-Cookie HTTP response header.

  Type: time.Duration
  Default: 2h0m0s
  Environment variable: TCM_HTTP_WEBSESSION_COOKIE_TTL
  Command-line option: `--http-websession-cookie-ttl`

`http.websession-cookie.secure`
: Indicates whether the cookie can be sent only over the HTTPS protocol. In this case, it's never sent over the unencrypted HTTP, therefore preventing man-in-the-middle attacks.

  When `true`, the `Secure` attribute is added to the Set-Cookie HTTP response header.

  Type: bool
  Default: false
  Environment variable: TCM_HTTP_WEBSESSION_COOKIE_SECURE
  Command-line option: `--http-websession-cookie-secure`

`http.websession-cookie.http-only`
: Indicates that the cookie can't be accessed from the JavaScript Document.cookie API. This helps mitigate cross-site scripting attacks.

  When `true`, the `HttpOnly` attribute is added to the Set-Cookie HTTP response header.

  Type: bool
  Default: true
  Environment variable: TCM_HTTP_WEBSESSION_COOKIE_HTTP_ONLY
  Command-line option: `--http-websession-cookie-http-only`

`http.websession-cookie.same-site`
: Indicates if it is possible to send the TCM cookie along with cross-site requests. Possible values are the Go's http.SameSite constants:

  - `SameSiteDefaultMode`
  - `SameSiteLaxMode`
  - `SameSiteStrictMode`
  - `SameSiteNoneMode`

  For details on `SameSite` modes, see the Set-Cookie header documentation in the MDN web docs.

  This value is used in the `SameSite` attribute of the Set-Cookie HTTP response header.

Type: http.SameSite
Default: SameSiteDefaultMode
Environment variable: TCM_HTTP_WEBSESSION_COOKIE_SAME_SITE
Command-line option: `--http-websession-cookie-same-site`

`http.cors.enabled`
Indicates whether to use the Cross-Origin Resource Sharing (*CORS*).

Type: bool
Default: false
Environment variable: TCM_HTTP_CORS_ENABLED
Command-line option: `--http-cors-enabled`

`http.cors.allowed-origins`
The origins with which the HTTP response can be shared, separated by semicolons.

The specified values are sent in the Access-Control-Allow-Origin HTTP response headers.

Type: []string
Default: []
Environment variable: TCM_HTTP_CORS_ALLOWED_ORIGINS
Command-line option: `--http-cors-allowed-origins`

`http.cors.allowed-methods`
HTTP request methods that are allowed when accessing a resource, separated by semicolons.

The specified values are sent in the Access-Control-Allow-Methods HTTP header of a response to a CORS preflight request.

Type: []string
Default: []
Environment variable: TCM_HTTP_CORS_ALLOWED_METHODS
Command-line option: `--http-cors-allowed-methods`

`http.cors.allowed-headers`
HTTP headers that are allowed during the actual request, separated by semicolons.

The specified values are sent in the Access-Control-Allow-Headers HTTP header of a response to a CORS preflight request.

Type: []string
Default: []
Environment variable: TCM_HTTP_CORS_ALLOWED_HEADERS
Command-line option: `--http-cors-allowed-headers`

`http.cors.exposed-headers`
Response headers that should be made available to scripts running in the browser, in response to a cross-origin request, separated by semicolons.

The specified values are sent in the Access-Control-Expose-Headers HTTP response headers.

Type: []string
Default: []
Environment variable: TCM_HTTP_CORS_EXPOSED_HEADERS
Command-line option: `--http-cors-exposed-headers`

`http.cors.allow-credentials`
Whether to expose the response to the frontend JavaScript code when the request's credentials mode is `include`.

When `true`, the Access-Control-Allow-Credentials HTTP response header is sent.

Type: bool
Default: false
Environment variable: TCM_HTTP_CORS_ALLOW_CREDENTIALS
Command-line option: `--http-cors-allow-credentials`

`http.cors.debug`
For debug purposes.

Type: bool
Default: false

`http.metrics-endpoint`
The HTTP endpoint for TCM metrics in the Prometheus format.

Type: string
Default: /metrics
Environment variable: TCM_HTTP_METRICS_ENDPOINT
Command-line option: `--http-metrics-endpoint`

`http.tls.enabled`
: Indicates whether TLS is enabled for client connections to TCM.

  Type: bool
  Default: false
  Environment variable: TCM_HTTP_TLS_ENABLED
  Command-line option: `--http-tls-enabled`

`http.tls.cert-file`
: A path to a TLS certificate file. Mandatory when TLS is enabled.

  Type: string
  Default: «»
  Environment variable: TCM_HTTP_TLS_CERT_FILE
  Command-line option: `--http-tls-cert-file`

`http.tls.key-file`
: A path to a TLS private key file. Mandatory when TLS is enabled.

  Type: string
  Default: «»
  Environment variable: TCM_HTTP_TLS_KEY_FILE
  Command-line option: `--http-tls-key-file`

`http.tls.server`
: The TSL server.

  Type: string
  Default: «»
  Environment variable: TCM_HTTP_TLS_SERVER
  Command-line option: `--http-tls-server`

`http.tls.min-version`
: The minimum version of the TLS protocol.

  Type: uint16
  Default: 0

Environment variable: TCM_HTTP_TLS_MIN_VERSION
Command-line option: `--http-tls-min-version`

`http.tls.max-version`
The maximum version of the TLS protocol.

Type: uint16
Default: 0
Environment variable: TCM_HTTP_TLS_MAX_VERSION
Command-line option: `--http-tls-max-version`

`http.tls.curve-preferences`
Elliptic curves that are used for TLS connections. Possible values are the Go's tls.CurveID constants:

- `CurveP256`
- `CurveP384`
- `CurveP521`
- `X25519`

Type: []tls.CurveID
Default: []
Environment variable: TCM_HTTP_TLS_CURVE_PREFERENCES
Command-line option: `--http-tls-curve-preferences`

`http.tls.cipher-suites`
Enabled TLS cipher suites. Possible values are the Golang tls.TLS_* constants.

Type: []uint16
Default: []
Environment variable: TCM_HTTP_TLS_CIPHER_SUITES
Command-line option: `--http-tls-cipher-suites`

`http.read-timeout`
A timeout for reading an incoming request.

Type: time.Duration
Default: 30s
Environment variable: TCM_HTTP_READ_TIMEOUT

Command-line option: `--http-read-timeout`

`http.read-header-timeout`
A timeout for reading headers of an incoming request.

Type: time.Duration
Default: 30s
Environment variable: TCM_HTTP_READ_HEADER_TIMEOUT
Command-line option: `--http-read-header-timeout`

`http.write-timeout`
A timeout for writing a response.

Type: time.Duration
Default: 30s
Environment variable: TCM_HTTP_WRITE_TIMEOUT
Command-line option: `--http-write-timeout`

`http.idle-timeout`
The timeout for idle connections.

Type: time.Duration
Default: 30s
Environment variable: TCM_HTTP_IDLE_TIMEOUT
Command-line option: `--http-idle-timeout`

`http.disable-general-options-handler`
Whether the client requests with the `OPTIONS` HTTP method are allowed.

Type: bool
Default: false
Environment variable: TCM_HTTP_DISABLE_GENERAL_OPTIONS_HANDLER
Command-line option: `--http-disable-general-options-handler`

`http.max-header-bytes`
The maximum size (in bytes) of a header in a client's request to TCM.

Type: int
Default: 0
Environment variable: TCM_HTTP_MAX_HEADER_BYTES
Command-line option: `--http-max-header-bytes`

`http.api-timeout`
The stateboard update timeout.

Type: time.Duration
Default: 8s
Environment variable: TCM_HTTP_API_TIMEOUT
Command-line option: `--http-api-timeout`

`http.api-update-interval`
The stateboard update interval.

Type: time.Duration
Default: 5s
Environment variable: TCM_HTTP_API_UPDATE_INTERVAL
Command-line option: `--http-api-update-interval`

`http.frontend-dir`
The directory with custom TCM frontend files (for development purposes).

Type: string
Default: «»
Environment variable: TCM_HTTP_FRONTEND_DIR
Command-line option: `--http-frontend-dir`

`http.show-stack-trace`
Whether error stack traces are shown in the web UI.

Type: bool
Default: true
Environment variable: TCM_HTTP_SHOW_STACK_TRACE
Command-line option: `--http-show-stack-trace`

`http.trace`
: Whether all query tracing information is written in logs.

  Type: bool
  Default: false
  Environment variable: TCM_HTTP_TRACE
  Command-line option: `--http-trace`

`http.max-static-size`
: The maximum size (in bytes) of a static content sent to TCM.

  Type: int
  Default: 104857600
  Environment variable: TCM_HTTP_MAX_STATIC_SIZE
  Command-line option: `--http-max-static-size`

`http.graphql.complexity`
: The maximum complexity of GraphQL queries that TCM processes. If this value is exceeded, TCM returns an error.

  Type: int
  Default: 40
  Environment variable: TCM_HTTP_GRAPHQL_COMPLEXITY
  Command-line option: `--http-graphql-complexity`

### log

The `log` section defines the TCM logging parameters.

- *log.default.add-source*
- *log.default.show-stack-trace*
- *log.default.level*
- *log.default.format*
- *log.default.output*
- *log.default.no-colorized*
- *log.default.file.name*
- *log.default.file.maxsize*
- *log.default.file.maxage*
- *log.default.file.maxbackups*

- *log.default.file.compress*
- *log.default.syslog.protocol*
- *log.default.syslog.output*
- *log.default.syslog.priority*
- *log.default.syslog.facility*
- *log.default.syslog.tag*
- *log.default.syslog.timeout*
- *log.outputs*

`log.default.add-source`
: Whether sources are added to the TCM log.

  Type: bool
  Default: false
  Environment variable: TCM_LOG_DEFAULT_ADD_SOURCE
  Command-line option: `--log-default-add-source`

`log.default.show-stack-trace`
: Whether stack traces are added to the TCM log.

  Type: bool
  Default: false
  Environment variable: TCM_LOG_DEFAULT_SHOW_STACK_TRACE
  Command-line option: `--log-default-show-stack-trace`

`log.default.level`
: The default TCM logging level.

  Possible values:

  - `VERBOSE`
  - `INFO`
  - `WARN`
  - `ALARM`

  Type: string
  Default: INFO
  Environment variable: TCM_LOG_DEFAULT_LEVEL
  Command-line option: `--log-default-level`

`log.default.format`
TCM log entries format.

Possible values:

- `struct`
- `json`

Type: string
Default: struct
Environment variable: TCM_LOG_DEFAULT_FORMAT
Command-line option: `--log-default-format`

`log.default.output`
The output used for TCM log.

Possible values:

- `stdout`
- `stderr`
- `file`
- `syslog`

Type: string
Default: stdout
Environment variable: TCM_LOG_DEFAULT_OUTPUT
Command-line option: `--log-default-output`

`log.default.no-colorized`
Whether the stdout log is not colorized.

Type: bool
Default: false
Environment variable: TCM_LOG_DEFAULT_NO_COLORIZED
Command-line option: `--log-default-no-colorized`

`log.default.file.name`
The name of the TCM log file.

Type: string
Default: «»

Environment variable: TCM_LOG_DEFAULT_FILE_NAME
Command-line option: `--log-default-file-name`

`log.default.file.maxsize`
The maximum size (in bytes) of the TCM log file.

Type: int
Default: 0
Environment variable: TCM_LOG_DEFAULT_FILE_MAXSIZE
Command-line option: `--log-default-file-maxsize`

`log.default.file.maxage`
The maximum age of a TCM log file, in days.

Type: int
Default: 0
Environment variable: TCM_LOG_DEFAULT_FILE_MAXAGE
Command-line option: `--log-default-file-maxage`

`log.default.file.maxbackups`
The maximum number of users in TCM.

Type: int
Default: 0
Environment variable: TCM_LOG_DEFAULT_FILE_MAXBACKUPS
Command-line option: `--log-default-file-maxbackups`

`log.default.file.compress`
Indicated that TCM compresses log files upon rotation.

Type: bool
Default: false
Environment variable: TCM_LOG_DEFAULT_FILE_COMPRESS
Command-line option: `--log-default-file-compress`

`log.default.syslog.protocol`
The network protocol used for connecting to the syslog server. Typically, it's `tcp`, `udp`, or `unix`. All possible values are listed in the Go's net.Dial documentation.

Type: string
Default: tcp
Environment variable: TCM_LOG_DEFAULT_SYSLOG_PROTOCOL
Command-line option: `--log-default-syslog-protocol`

`log.default.syslog.output`
The syslog server URI.

Type: string
Default: 127.0.0.1:5514
Environment variable: TCM_LOG_DEFAULT_SYSLOG_OUTPUT
Command-line option: `--log-default-syslog-output`

`log.default.syslog.priority`
The syslog severity level.

Type: string
Default: «»
Environment variable: TCM_LOG_DEFAULT_SYSLOG_PRIORITY
Command-line option: `--log-default-syslog-priority`

`log.default.syslog.facility`
The syslog facility.

Type: string
Default: «»
Environment variable: TCM_LOG_DEFAULT_SYSLOG_FACILITY
Command-line option: `--log-default-syslog-facility`

`log.default.syslog.tag`
The syslog tag.

Type: string
Default: «»
Environment variable: TCM_LOG_DEFAULT_SYSLOG_TAG
Command-line option: `--log-default-syslog-tag`

`log.default.syslog.timeout`
: The timeout for connecting to the syslog server.

  Type: time.Duration
  Default: 10s
  Environment variable: TCM_LOG_DEFAULT_SYSLOG_TIMEOUT
  Command-line option: `--log-default-syslog-timeout`

`log.outputs`
: An array of log outputs that TCM uses **in addition** to the default one that is defined by the `log.default.*` parameters. Each array item can include the parameters of the `log.default` group. If a parameter is skipped, its value is taken from `log.default`.

  Type: []LogOuputConfig
  Default: []
  Environment variable: TCM_LOG_OUTPUTS
  Command-line option: `--log-outputs`

#### storage

The `storage` section defines the parameters of the configuration storage that TCM uses for connected clusters.

- *storage.provider*

etcd storage parameters:

- *storage.etcd.prefix*
- *storage.etcd.endpoints*
- *storage.etcd.dial-timeout*
- *storage.etcd.auto-sync-interval*
- *storage.etcd.dial-keep-alive-time*
- *storage.etcd.dial-keep-alive-timeout*
- *storage.etcd.bootstrap-timeout*
- *storage.etcd.max-call-send-msg-size*
- *storage.etcd.username*
- *storage.etcd.password*
- *storage.etcd.tls.enabled*
- *storage.etcd.tls.auto*
- *storage.etcd.tls.cert-file*
- *storage.etcd.tls.key-file*

- *storage.etcd.tls.trusted-ca-file*
- *storage.etcd.tls.client-cert-auth*
- *storage.etcd.tls.crl-file*
- *storage.etcd.tls.insecure-skip-verify*
- *storage.etcd.tls.skip-client-san-verify*
- *storage.etcd.tls.server-name*
- *storage.etcd.tls.cipher-suites*
- *storage.etcd.tls.allowed-cn*
- *storage.etcd.tls.allowed-hostname*
- *storage.etcd.tls.empty-cn*
- *storage.etcd.permit-without-stream*
- *storage.etcd.embed.enabled*
- *storage.etcd.embed.endpoints*
- *storage.etcd.embed.advertises*
- *storage.etcd.embed.tls.enabled*
- *storage.etcd.embed.tls.auto*
- *storage.etcd.embed.tls.cert-file*
- *storage.etcd.embed.tls.key-file*
- *storage.etcd.embed.tls.trusted-ca-file*
- *storage.etcd.embed.tls.client-cert-auth*
- *storage.etcd.embed.tls.crl-file*
- *storage.etcd.embed.tls.insecure-skip-verify*
- *storage.etcd.embed.tls.skip-client-san-verify*
- *storage.etcd.embed.tls.server-name*
- *storage.etcd.embed.tls.cipher-suites*
- *storage.etcd.embed.tls.allowed-cn*
- *storage.etcd.embed.tls.allowed-hostname*
- *storage.etcd.embed.tls.empty-cn*
- *storage.etcd.embed.peer-endpoints*
- *storage.etcd.embed.peer-advertises*
- *storage.etcd.embed.peer-tls.enabled*
- *storage.etcd.embed.peer-tls.auto*
- *storage.etcd.embed.peer-tls.cert-file*
- *storage.etcd.embed.peer-tls.key-file*
- *storage.etcd.embed.peer-tls.trusted-ca-file*
- *storage.etcd.embed.peer-tls.client-cert-auth*

- *storage.etcd.embed.peer-tls.crl-file*
- *storage.etcd.embed.peer-tls.insecure-skip-verify*
- *storage.etcd.embed.peer-tls.skip-client-san-verify*
- *storage.etcd.embed.peer-tls.server-name*
- *storage.etcd.embed.peer-tls.cipher-suites*
- *storage.etcd.embed.peer-tls.allowed-cn*
- *storage.etcd.embed.peer-tls.allowed-hostname*
- *storage.etcd.embed.peer-tls.empty-cn*
- *storage.etcd.embed.grpc-keep-alive-timeout*
- *storage.etcd.embed.grpc-keep-alive-interval*
- *storage.etcd.embed.grpc-keep-alive-min-time*
- *storage.etcd.embed.workdir*
- *storage.etcd.embed.waldir*
- *storage.etcd.embed.max-request-bytes*
- *storage.etcd.embed.debug*
- *storage.etcd.embed.start-timeout*
- *storage.etcd.embed.log-level*
- *storage.etcd.embed.initial-cluster*
- *storage.etcd.embed.initial-cluster-token*
- *storage.etcd.embed.name*
- *storage.etcd.embed.initial-cluster-state*
- *storage.etcd.embed.self-signed-cert-validity*

Tarantool storage parameters:

- *storage.tarantool.prefix*
- *storage.tarantool.addr*
- *storage.tarantool.auth*
- *storage.tarantool.reconnect*
- *storage.tarantool.max-reconnects*
- *storage.tarantool.user*
- *storage.tarantool.pass*
- *storage.tarantool.rate-limit*
- *storage.tarantool.rate-limit-action*
- *storage.tarantool.concurrency*
- *storage.tarantool.skip-schema*
- *storage.tarantool.transport*
- *storage.tarantool.ssl.key-file*

- *storage.tarantool.ssl.cert-file*
- *storage.tarantool.ssl.ca-file*
- *storage.tarantool.ssl.ciphers*
- *storage.tarantool.ssl.password*
- *storage.tarantool.required-protocol-info.auth*
- *storage.tarantool.required-protocol-info.version*
- *storage.tarantool.required-protocol-info.features*
- *storage.tarantool.embed.enabled*
- *storage.tarantool.embed.workdir*
- *storage.tarantool.embed.executable*
- *storage.tarantool.embed.config-filename*
- *storage.tarantool.embed.config*
- *storage.tarantool.embed.args*
- *storage.tarantool.embed.env*

`storage.provider`
The type of the storage used for storing TCM configuration.

Possible values:

- `etcd`
- `tarantool`

Type: string
Default: etcd
Environment variable: TCM_STORAGE_PROVIDER
Command-line option: `--storage-provider`

`storage.etcd.prefix`
A prefix for the TCM configuration parameters in etcd.

Type: string
Default: «/tcm»
Environment variable: TCM_STORAGE_ETCD_PREFIX
Command-line option: `--storage-etcd-prefix`

`storage.etcd.endpoints`
An array of node URIs of the etcd cluster where the TCM configuration is stored, separated by semicolons (`;`).

Type: []string
Default: [«http://127.0.0.1:2379»]
Environment variable: TCM_STORAGE_ETCD_ENDPOINTS
Command-line option: `--storage-etcd-endpoints`

`storage.etcd.dial-timeout`
An etcd dial timeout.

Type: time.Duration
Default: 10s
Environment variable: TCM_STORAGE_ETCD_DIAL_TIMEOUT
Command-line option: `--storage-etcd-dial-timeout`

`storage.etcd.auto-sync-interval`
An automated sync interval.

Type: time.Duration
Default: 0s
Environment variable: TCM_STORAGE_ETCD_AUTO_SYNC_INTERVAL
Command-line option: `--storage-etcd-auto-sync-interval`

`storage.etcd.dial-keep-alive-time`
A dial keep-alive time.

Type: time.Duration
Default: 30s
Environment variable: TCM_STORAGE_ETCD_DIAL_KEEP_ALIVE_TIME
Command-line option: `--storage-etcd-dial-keep-alive-time`

`storage.etcd.dial-keep-alive-timeout`
A dial keep-alive timeout.

Type: time.Duration
Default: 30s
Environment variable: TCM_STORAGE_ETCD_DIAL_KEEP_ALIVE_TIMEOUT
Command-line option: `--storage-etcd-dial-keep-alive-timeout`

`storage.etcd.bootstrap-timeout`
A bootstrap timeout.

Type: time.Duration
Default: 30s
Environment variable: TCM_STORAGE_ETCD_BOOTSTRAP_TIMEOUT
Command-line option: `--storage-etcd-bootstrap-timeout`

`storage.etcd.max-call-send-msg-size`
The maximum size (in bytes) of a transaction between TCM and etcd.

Type: int
Default: 2097152
Environment variable: TCM_STORAGE_ETCD_MAX_CALL_SEND_MSG_SIZE
Command-line option: `--storage-etcd-max-call-send-msg-size`

`storage.etcd.username`
A username for accessing the etcd storage.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_USERNAME
Command-line option: `--storage-etcd-username`

`storage.etcd.password`
A password for accessing the etcd storage.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_PASSWORD
Command-line option: `--storage-etcd-password`

`storage.etcd.tls.enabled`
Indicates whether TLS is enabled for etcd connections.

Type: bool
Default: false

Environment variable: TCM_STORAGE_ETCD_TLS_ENABLED
Command-line option: `--storage-etcd-tls-enabled`

`storage.etcd.tls.auto`
Use generated certificates for etcd connections.

Type: bool
Default: false
Environment variable: TCM_STORAGE_ETCD_TLS_AUTO
Command-line option: `--storage-etcd-tls-auto`

`storage.etcd.tls.cert-file`
A path to a TLS certificate file to use for etcd connections.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_TLS_CERT_FILE
Command-line option: `--storage-etcd-tls-cert-file`

`storage.etcd.tls.key-file`
A path to a TLS private key file to use for etcd connections.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_TLS_KEY_FILE
Command-line option: `--storage-etcd-tls-key-file`

`storage.etcd.tls.trusted-ca-file`
A path to a trusted CA certificate file to use for etcd connections.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_TLS_TRUSTED_CA_FILE
Command-line option: `--storage-etcd-tls-trusted-ca-file`

`storage.etcd.tls.client-cert-auth`
Indicates whether client cert authentication is enabled.

Type: bool
Default: false
Environment variable: TCM_STORAGE_ETCD_TLS_CLIENT_CERT_AUTH
Command-line option: `--storage-etcd-tls-client-cert-auth`

`storage.etcd.tls.crl-file`
A path to the client certificate revocation list file.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_TLS_CRL_FILE
Command-line option: `--storage-etcd-tls-crl-file`

`storage.etcd.tls.insecure-skip-verify`
Skip checking client certificate in etcd connections.

Type: bool
Default: false
Environment variable: TCM_STORAGE_ETCD_TLS_INSECURE_SKIP_VERIFY
Command-line option: `--storage-etcd-tls-insecure-skip-verify`

`storage.etcd.tls.skip-client-san-verify`
Skip verification of SAN field in client certificate for etcd connections.

Type: bool
Default: false
Environment variable: TCM_STORAGE_ETCD_TLS_SKIP_CLIENT_SAN_VERIFY
Command-line option: `--storage-etcd-tls-skip-client-san-verify`

`storage.etcd.tls.server-name`
Name of the TLS server for etcd connections.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_TLS_SERVER_NAME
Command-line option: `--storage-etcd-tls-server-name`

`storage.etcd.tls.cipher-suites`
TLS cipher suites for etcd connections. Possible values are the Golang tls.TLS_* constants.

Type: []uint16
Default: []
Environment variable: TCM_STORAGE_ETCD_TLS_CIPHER_SUITES
Command-line option: `--storage-etcd-tls-cipher-suites`

`storage.etcd.tls.allowed-cn`
An allowed common name for authentication in etcd connections.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_TLS_ALLOWED_CN
Command-line option: `--storage-etcd-tls-allowed-cn`

`storage.etcd.tls.allowed-hostname`
An allowed TLS certificate name for authentication in etcd connections.

Type: string
Default: «»
Environment variable: TCM_STORAGE_ETCD_TLS_ALLOWED_HOSTNAME
Command-line option: `--storage-etcd-tls-allowed-hostname`

`storage.etcd.tls.empty-cn`
Whether the empty common name is allowed in etcd connections.

Type: bool
Default: false
Environment variable: TCM_STORAGE_ETCD_TLS_EMPTY_CN
Command-line option: `--storage-etcd-tls-empty-cn`

`storage.etcd.permit-without-stream`
Whether keepalive pings can be send to the etcd server without active streams.

Type: bool
Default: false

Environment variable: TCM_STORAGE_ETCD_PERMIT_WITHOUT_STREAM
Command-line option: `--storage-etcd-permit-without-stream`

### storage.etcd.embed.*

The `storage.etcd.embed` group defines the configuration of the embedded etcd cluster that can used as a TCM configuration storage. This cluster can be used for development purposes when the production or testing etcd cluster is not available or not needed.

`storage.tarantool.prefix`
A prefix for the TCM configuration parameters in the Tarantool TCM configuration storage.

Type: string
Default: «_tcm:
Environment variable: TCM_STORAGE_TARANTOOL_PREFIX
Command-line option: `--storage-tarantool-prefix`

`storage.tarantool.addr`
The URI for connecting to the Tarantool TCM configuration storage.

Type: string
Default: «unix/:/tmp/tnt_config_instance.sock»
Environment variable: TCM_STORAGE_TARANTOOL_ADDR
Command-line option: `--storage-tarantool-ADDR`

`storage.tarantool.auth`
An authentication method for the Tarantool TCM configuration storage.

Possible values are the Go's go-tarantool/Auth constants:

- `AutoAuth` (0)
- `ChapSha1Auth`
- `PapSha256Auth`

Type: int
Default: 0
Environment variable: TCM_STORAGE_TARANTOOL_AUTH
Command-line option: `--storage-tarantool-auth`

`storage.tarantool.timeout`
A request timeout for the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: time.Duration
Default: 0s
Environment variable: TCM_STORAGE_TARANTOOL_TIMEOUT
Command-line option: `--storage-tarantool-timeout`

`storage.tarantool.reconnect`
A timeout between reconnect attempts for the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: time.Duration
Default: 0s
Environment variable: TCM_STORAGE_TARANTOOL_RECONNECT
Command-line option: `--storage-tarantool-reconnect`

`storage.tarantool.max-reconnects`
The maximum number of reconnect attempts for the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: int
Default: 0
Environment variable: TCM_STORAGE_TARANTOOL_MAX_RECONNECTS
Command-line option: `--storage-tarantool-max-reconnects`

`storage.tarantool.user`
A username for connecting to the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: string
Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_USER
Command-line option: `--storage-tarantool-user`

`storage.tarantool.pass`
A password for connecting to the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: string
Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_PASS
Command-line option: `--storage-tarantool-pass`

`storage.tarantool.rate-limit`
A rate limit for connecting to the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: int
Default: 0
Environment variable: TCM_STORAGE_TARANTOOL_RATE_LIMIT
Command-line option: `--storage-tarantool-rate-limit`

`storage.tarantool.rate-limit-action`
An action to perform when the tcm_configuration_reference_storage_tarantool_rate-limit is reached.

See also go-tarantool.Opts.

Type: int
Default: 0
Environment variable: TCM_STORAGE_TARANTOOL_RATE_LIMIT_ACTION
Command-line option: `--storage-tarantool-rate-limit-action`

`storage.tarantool.concurrency`
An amount of separate mutexes for request queues and buffers inside of a connection to the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: int
Default: 0
Environment variable: TCM_STORAGE_TARANTOOL_CONCURRENCY
Command-line option: `--storage-tarantool-concurrency`

`storage.tarantool.skip-schema`
Whether the schema is loaded from the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: bool
Default: true
Environment variable: TCM_STORAGE_TARANTOOL_SKIP_SCHEMA
Command-line option: `--storage-tarantool-skip-schema`

`storage.tarantool.transport`
The connection type for the Tarantool TCM configuration storage.

See also go-tarantool.Opts.

Type: string
Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_TRANSPORT
Command-line option: `--storage-tarantool-transport`

`storage.tarantool.ssl.key-file`
A path to a TLS private key file to use for connecting to the Tarantool TCM configuration storage.

See also: *Securing connections with SSL*.

Type: string
Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_SSL_KEY_FILE
Command-line option: `--storage-tarantool-ssl-key-file`

`storage.tarantool.ssl.cert-file`
A path to an SSL certificate to use for connecting to the Tarantool TCM configuration storage.

See also: *Securing connections with SSL*.

Type: string
Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_SSL_CERT_FILE
Command-line option: `--storage-tarantool-ssl-cert-file`

`storage.tarantool.ssl.ca-file`
A path to a trusted CA certificate to use for connecting to the Tarantool TCM configuration storage.

See also: *Securing connections with SSL*.

Type: string

Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_SSL_CA_FILE
Command-line option: `--storage-tarantool-ssl-ca-file`

`storage.tarantool.ssl.ciphers`
A list of SSL cipher suites that can be used for connecting to the Tarantool TCM configuration storage. Possible values are listed in *<uri>.params.ssl_ciphers*.

See also: *Securing connections with SSL*.

Type: string
Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_SSL_CIPHERS
Command-line option: `--storage-tarantool-ssl-ciphers`

`storage.tarantool.ssl.password`
A password for an encrypted private SSL key to use for connecting to the Tarantool TCM configuration storage.

See also: *Securing connections with SSL*.

Type: string
Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_SSL_PASSWORD
Command-line option: `--storage-tarantool-ssl-password`

`storage.tarantool.ssl.password-file`
A text file with passwords for encrypted private SSL keys to use for connecting to the Tarantool TCM configuration storage.

Type: string
Default: «»
Environment variable: TCM_STORAGE_TARANTOOL_SSL_PASSWORD_FILE
Command-line option: `--storage-tarantool-ssl-password-file`

`storage.tarantool.required-protocol-info.auth`
An authentication method for the Tarantool TCM configuration storage.

Possible values are the Go's go-tarantool/Auth constants:

- `AutoAuth` (0)
- `ChapSha1Auth`
- `PapSha256Auth`

See also go-tarantool.ProtocolInfo.

Type: int
Default: 0
Environment variable:
TCM_STORAGE_TARANTOOL_SSL_REQUIRED_PROTOCOL_INFO_AUTH
Command-line option: `--storage-tarantool-required-protocol-info-auth`

`storage.tarantool.required-protocol-info.version`
A Tarantool protocol version.

See also go-tarantool.ProtocolInfo.

Type: uint64
Default: 0
Environment variable:
TCM_STORAGE_TARANTOOL_SSL_REQUIRED_PROTOCOL_INFO_VERSION
Command-line option: `--storage-tarantool-required-protocol-info-version`

`storage.tarantool.required-protocol-info.features`
An array of Tarantool protocol features.

See also go-tarantool.ProtocolInfo.

Type: []int
Default: []
Environment variable:
TCM_STORAGE_TARANTOOL_SSL_REQUIRED_PROTOCOL_INFO_FEATURES
Command-line option: `--storage-tarantool-required-protocol-info-features`

### storage.tarantool.embed.*

The `storage.tarantool.embed` group parameters define the configuration of the embedded Tarantool cluster that can used as a TCM configuration storage. This cluster can be used for development purposes when the production or testing cluster is not available or not needed.

### addon

The `addon` section defines settings related to TCM add-ons.

- *addon.enabled*
- *addon.addons-dir*
- *addon.max-upload-size*
- *addon.dev-addons-dir*

`addon.enabled`
: Whether to enable the add-on functionality in TCM.

  Type: bool
  Default: false
  Environment variable: TCM_ADDON_ENABLED
  Command-line option: `--addon-enabled`

`addon.addons-dir`
: The directory from which TCM takes add-ons.

  Type: string
  Default: addons
  Environment variable: TCM_ADDON_ADDONS_DIR
  Command-line option: `--addon-addons-dir`

`addon.max-upload-size`
: The maximum size (in bytes) of addon to upload to TCM.

  Type: int64
  Default: 104857600
  Environment variable: TCM_ADDON_MAX_UPLOAD_SIZE
  Command-line option: `--addon-max-upload-size`

`addon.dev-addons-dir`
: Additional add-on directories for development purposes, separated by semicolons (`;`).

  Type: []string
  Default: []
  Environment variable: TCM_ADDON_DEV_ADDONS_DIR
  Command-line option: `--addon-dev-addons-dir`

### limits

The `limits` section defines limits on various TCM objects and relations between them.

- *limits.users-count*
- *limits.clusters-count*
- *limits.roles-count*
- *limits.user-secrets-count*
- *limits.user-websessions-count*
- *limits.linked-cluster-users*

`limits.users-count`
: The maximum number of users in TCM.

  Type: int
  Default: 1000
  Environment variable: TCM_LIMITS_USERS_COUNT
  Command-line option: `--limits-users-count`

`limits.clusters-count`
: The maximum number of clusters in TCM.

  Type: int
  Default: 10
  Environment variable: TCM_LIMITS_CLUSTERS_COUNT
  Command-line option: `--limits-clusters-count`

`limits.roles-count`
: The maximum number of roles in TCM.

  Type: int
  Default: 100
  Environment variable: TCM_LIMITS_ROLES_COUNT
  Command-line option: `--limits-roles-count`

`limits.user-secrets-count`
: The maximum number secrets that a TCM user can have.

  Type: int

Default: 10
Environment variable: TCM_LIMITS_USER_SECRETS_COUNT
Command-line option: `--limits-user-secrets-count`

`limits.user-websessions-count`
The maximum number of open sessions that a TCM user can have.

Type: int
Default: 10
Environment variable: TCM_LIMITS_USER_WEBSESSIONS_COUNT
Command-line option: `--limits-user-websessions-count`

`limits.linked-cluster-users`
The maximum number of clusters to which a single user can have access.

Type: int
Default: 10
Environment variable: TCM_LIMITS_LINKED_CLUSTER_USERS
Command-line option: `--limits-linked-cluster-users`

### security

The `security` section defines the security parameters of TCM.

- *security.auth*
- *security.hash-cost*
- *security.encryption-key*
- security.encryption-key-file
- *security.bootstrap-password*
- security.integrity-check
- security.signature-private-key-file

`security.auth`
Ways to log into TCM.

Possible values:

- `local`
- `ldap`

Type: []string

Default: [local]
Environment variable: TCM_SECURITY_AUTH
Command-line option: `--security-auth`

`security.hash-cost`
A hash cost for hashing users“ passwords.

Type: int
Default: 12
Environment variable: TCM_SECURITY_HASH_COST
Command-line option: `--security-hash-cost`

`security.encryption-key`
An encryption key for passwords used by TCM for accessing Tarantool and etcd clusters.

Type: string
Default: «»
Environment variable: TCM_SECURITY_ENCRYPTION_KEY
Command-line option: `--security-encryption-key`

`security.encryption-key-file`
A path to the file with the encryption key for passwords used by TCM for accessing Tarantool and etcd clusters.

Type: string
Default: «»
Environment variable: TCM_SECURITY_ENCRYPTION_KEY_FILE
Command-line option: `--security-encryption-key-file`

`security.bootstrap-password`
A password for the first login of the `admin` user. Must be changed after the successful login. Only for testing purposes.

Type: string
Default: «»
Environment variable: TCM_SECURITY_BOOTSTRAP_PASSWORD
Command-line option: `--security-bootstrap-password`

`security.signature-private-key-file`
A path to a file with the private key to sign TCM data.

Type: string
Default: «»
Environment variable: TCM_SECURITY_SIGNATURE_PRIVATE_KEY_FILE
Command-line option: `--security-signature-private-key-file`

`security.integrity-check`
Whether to check the digital signature. If `true`, the error is raised in case an incorrect signature is detected.

Type: bool
Default: false
Environment variable: TCM_SECURITY_INTEGRITY_CHECK
Command-line option: `--security-integrity-check`

**mode**

`mode`
The TCM mode: `production`, `development`, or `test`.

Type: string
Default: production
Environment variable: TCM_MODE
Command-line option: `--mode`

### 9.3.3 Interactive console

The interactive console is Tarantool's basic command-line interface for entering requests and seeing results. It is what users see when they start the server without an *instance file*. The interactive console is often called the Lua console to distinguish it from the administrative console, but in fact it can handle both Lua and SQL input.

The majority of examples in this manual show what users see with the interactive console. It includes:

- `tarantool>` prompt
- instruction (a Lua request or an SQL statement)
- response (a display in either YAML or Lua format)

```
-- Interactive console example with Lua input and YAML output --
tarantool> box.info().replication
---
- 1:
    id: 1
    uuid: a5d22f66-2d28-4a35-b78f-5bf73baf6c8a
    lsn: 0
...
```

### Interactive console input and output

The **input language** can be either Lua (default) or SQL. To change the input language, run `\set language <language>`, for example:

```
-- Set input language to SQL --
tarantool> \set language sql
---
- true
...
```

The **delimiter** can be changed to any character with `\set delimiter <character>`. By default, the delimiter is empty, which means the input does not need to end with a delimiter. For example, a common recommendation for SQL input is to use the semicolon delimiter:

```
-- Set ';' delimiter --
tarantool> \set delimiter ;
---
...
```

The **output format** can be either YAML (default) or Lua. To change the output format, run `\set output <format>`, for example:

```
-- Set output format Lua --
tarantool> \set output lua
true
```

The default YAML output format is the following:

- The output starts from a document-start line `"---"`.
- Each item begins on a separate line starting with `"- "`.
- Each sub-item in a nested structure is indented.
- The output ends with a document-end line `"..."`.

The alternative Lua format for console output is the following:

- There are no lines for document-start or document-end.
- Items are separated by commas.
- Each sub-item in a nested structure is placed inside «`{}`» braces.

So, when an input is a Lua object description, the output in the Lua format equals it.

For the Lua output format, you can specify an **end of statement** symbol. It is added to the end of each output statement in the current session and can be used for parsing the output by scripts. By default, the

end of statement symbol is empty. You can change it to any character or character sequence. To set an end of statement symbol for the current session, run \`set output lua,local_eos=<symbol>`, for example:

```
-- Set output format Lua and '#' end of statement symbol --
tarantool> \set output lua,local_eos=#
true#
```

To switch back to the empty end of statement symbol:

```
-- Set output format Lua and empty end of statement symbol --
tarantool> \set output lua,local_eos=
true
```

The YAML output has better readability. The Lua output can be reused in requests. The table below shows output examples in these formats compared with the MsgPack format, which is good for database storage.

| Тип | Lua input | Lua output | YAML output | MsgPack storage |
|---|---|---|---|---|
| скалярный | `1` | `1` | `---`<br>`- 1`<br>`...` | `\x01` |
| scalar sequence | `1, 2, 3` | `1, 2, 3` | `---`<br>`- 1`<br>`- 2`<br>`- 3`<br>`...` | `\x01 \x02 \x03` |
| 2-element table | `{1, 2}` | `{1, 2}` | `---`<br>`- - 1`<br>`  - 2`<br>`...` | `0x92 0x01 0x02` |
| map (ассоциативный массив) | `{key = 1}` | `{key = 1}` | `---`<br>`- key: 1`<br>`...` | `\x81 \xa3 \x6b \x65 \x79 \ x01` |

The console parameters of a Tarantool instance can also be changed from another instance using the *console* built-in module functions.

#### Keyboard shortcuts

Since *2.10.0*.

| Keyboard shortcut | Effect |
|---|---|
| `CTRL+C` | Discard current input with the `SIGINT` signal in the console mode and jump to a new line with a default prompt. |
| `CTRL+D` | Quit Tarantool interactive console. |

**Важно:** Keep in mind that `CTRL+C` shortcut will shut Tarantool down if there is any currently running command in the console. The *SIGINT* signal stops the instance running in a daemon mode.

### 9.3.4 tarantoolctl utility (deprecated)

**Важно:** `tarantoolctl` is deprecated in favor of *tt CLI*. Find the instructions on switching from `tarantoolctl` to `tt` in *Migration from tarantoolctl to tt*.

`tarantoolctl` is a utility for administering Tarantool *instances*, *checkpoint files* and *modules*. It is shipped and installed as part of Tarantool distribution. This utility is intended for use by administrators only.

См. также примеры использования `tarantoolctl` в разделе *Администрирование серверной части*.

#### Формат команд

```
tarantoolctl COMMAND NAME [URI] [FILE] [OPTIONS..]
```

где:

- `COMMAND` – это одна из следующих команд, описанных ниже: `start`, `stop`, `status`, `restart`, `logrotate`, `check`, `enter`, `eval`, `connect`, `cat`, `play`, `rocks`.
- `NAME` – это имя *файла экземпляра* или *модуля*.
- `FILE` – это путь к какому-либо файлу (.lua, .xlog или .snap).
- `URI` – это URI некого экземпляра Tarantool.
- `OPTIONS` – это параметры, которые принимают команды `tarantoolctl`.

#### Команды для управления экземплярами Tarantool

`tarantoolctl start NAME` Запуск экземпляра Tarantool'а.

Кроме того, данная команда задает значение переменной окружения TARANTOOLCTL = „true“ (правда), чтобы отметить, что экземпляр был запущен с помощью `tarantoolctl`.

**Примечание:** `tarantoolctl` работает для экземпляров, где не вызвана функция `box.cfg{}` или вызов `box.cfg{}` отложен.

Например, это можно использовать для управления экземплярами, которые получают конфигурацию из внешнего сервера. Для таких экземпляров `tarantoolctl start` goes to background

when `box.cfg{}` is called, so it will wait until options for `box.cfg` are received. However this is not the case for daemon management systems like `systemd`, as they handle backgrounding on their side.

---

`tarantoolctl stop NAME` Остановка экземпляра Tarantool.

`tarantoolctl status NAME` Отображение статуса экземпляра (работает/остановлен). Если есть PID-файл и активный управляющий сокет, возвращается код `0`. В остальных случаях возвращается не `0`.

Сообщает о типичных проблемах стандартного вывода ошибок (например, PID-файл есть, а управляющий сокет отсутствует).

`tarantoolctl restart NAME` Остановка и запуск экземпляра Tarantool'а.

Кроме того, данная команда задает значение переменной окружения TARANTOOL_RESTARTED = „true“ (правда), чтобы отметить, что экземпляр был перезапущен с помощью `tarantoolctl`.

`tarantoolctl logrotate NAME` Ротация файлов журнала работающего Tarantool-экземпляра. Работает только в том случае, если в файле экземпляра задан параметр записи журнала в файл. Отправка записей в конвейер или системный журнал syslog не имеет значения в данном случае.

`tarantoolctl check NAME` Проверка файла экземпляра на ошибки синтаксиса.

`tarantoolctl enter NAME [--language=language]` Enter an instance's interactive Lua or SQL console.

Supported option:

- `--language=language` to set *interactive console* language. Can be either `Lua` or `SQL`.

`tarantoolctl eval NAME FILE` Выполнение локального Lua-файла на работающем экземпляре Tarantool.

`tarantoolctl connect URI` Подключение к экземпляру Tarantool по порту административной консоли. Поддерживаются TCP и Unix сокеты.

### Команды для управления файлами контрольной точки

`tarantoolctl cat FILE.. [--space=space_no ..] [--show-system] [--from=from_lsn] [--to=to_lsn] [--replica`
Стандартный вывод содержимого .snap-файла или .xlog-файла.

`tarantoolctl play URI FILE.. [--space=space_no ..] [--show-system] [--from=from_lsn] [--to=to_lsn] [--r`
Передача содержимого .snap-файла или .xlog-файла на другой экземпляр Tarantool.

Поддерживаемые опции:

- `--space=space_no` для фильтрации вывода по номеру спейса. Можно передавать несколько раз.
- `--show-system` для отображения содержимого системных спейсов.
- `--from=from_lsn` для отображения операций, начиная с заданного LSN.
- `--to=to_lsn` для отображения операций, заканчивая заданным LSN.
- `--replica=replica_id` для фильтрации вывода по идентификатору реплики. Можно передавать несколько раз.
- `--format=format_name` для указания формата (по умолчанию `yaml`, может также принимать значения `json` и `lua`).

### Команды для управления модулями Tarantool

`tarantoolctl rocks build NAME` Build/compile and install a rock. Since version *2.4.1*.

`tarantoolctl rocks config URI` Query and set the LuaRocks configuration. Since version *2.4.1*.

`tarantoolctl rocks doc NAME` Отображение документации для установленного модуля.

`tarantoolctl rocks download [NAME]` Download a specific rock or rockspec file from a rocks server. Since version *2.4.1*.

`tarantoolctl rocks help NAME` Помощь по командам.

`tarantoolctl rocks init NAME` Initialize a directory for a Lua project using LuaRocks. Since version *2.4.1*.

`tarantoolctl rocks install NAME` Установка модуля в директорию `.rocks`, находящуюся в текущей директории.

`tarantoolctl rocks lint FILE` Check the syntax of a rockspec. Since version *2.4.1*.

`tarantoolctl rocks list` Вывод списка всех установленных модулей.

`tarantoolctl rocks make` Компиляция пакета в текущем каталоге с помощью rockspec и его установка.

`tarantoolctl rocks make_manifest` Компиляция файла-манифеста для репозитория.

`tarantoolctl rocks new_version NAME` Auto-write a rockspec for a new version of a rock. Since version *2.4.1*.

`tarantoolctl rocks pack NAME` Создание модуля путем компоновки исходных или бинарных файлов.

`tarantoolctl rocks purge NAME` Remove all installed rocks from a tree. Since version *2.4.1*.

`tarantoolctl rocks remove NAME` Удаление модуля.

`tarantoolctl rocks show NAME` Отображение информации об установленном модуле.

`tarantoolctl rocks search NAME` Поиск модулей по репозиторию.

`tarantoolctl rocks unpack NAME` Распаковка содержимого модуля.

`tarantoolctl rocks which NAME` Tell which file corresponds to a given module name. Since version *2.4.1*.

`tarantoolctl rocks write_rockspec` Write a template for a rockspec file. Since version *2.4.1*.

В качестве аргумента можно указать:

- файл в формате `.rockspec` для создания модуля, который содержит исходные файлы или
- имя установленного модуля (с версией, если их больше одной) для создания модуля, который содержит скомпилированные файлы.

`tarantoolctl rocks unpack {<rock_file> | <rockspec> | <имя> [версия]}` Распаковка содержимого модуля в новую директорию в текущей директории.

В качестве аргумента можно указать:

- исходные или бинарные файлы модуля,
- файлы `.rockspec` или
- имя модулей или файлов в формате `.rockspec` в удаленных репозиториях (с версией модуля, если их больше одной).

Поддерживаемые опции:

- `--server=имя_севрера` сначала проверить данный сервер, затем по списку.
- `--only-server=имя_сервера` проверить только данный сервер, остальные пропустить.

### tarantoolctl configuration file

The `tarantoolctl` configuration file named `.tarantoolctl` contains the configuration that `tarantoolctl` uses to override instance configuration. In other words, it contains system-wide configuration defaults. If `tarantoolctl` fails to find this file with the method described in the section *Starting/stopping an instance*, it uses default settings.

Most of the parameters are similar to those used by *box.cfg{}*. Here are the default settings (possibly installed in `/etc/default/tarantool` or `/etc/sysconfig/tarantool` as part of Tarantool distribution – see OS-specific default paths in *Notes for operating systems*):

```
default_cfg = {
    pid_file  = "/var/run/tarantool",
    wal_dir   = "/var/lib/tarantool",
    memtx_dir = "/var/lib/tarantool",
    vinyl_dir = "/var/lib/tarantool",
    log       = "/var/log/tarantool",
    username  = "tarantool",
    language  = "Lua",
}
instance_dir = "/etc/tarantool/instances.enabled"
```

где:

- `pid_file`
  Directory for the pid file and control-socket file; `tarantoolctl` will add “/instance_name” to the directory name.
- `wal_dir`
  Directory for write-ahead .xlog files; `tarantoolctl` will add «/instance_name» to the directory name.
- `memtx_dir`
  Directory for snapshot .snap files; `tarantoolctl` will add «/instance_name» to the directory name.
- `vinyl_dir`
  Directory for vinyl files; `tarantoolctl` will add «/instance_name» to the directory name.
- `log`
  The place where the application log will go; `tarantoolctl` will add «/instance_name.log» to the name.
- `username`
  The user that runs the Tarantool instance. This is the operating system user name rather than the Tarantool-client user name. Tarantool will change its effective user to this user after becoming a daemon.
- `language`
  The *interactive console* language. Can be either `Lua` or `SQL`.
- `instance_dir`
  The directory where all instance files for this host are stored. Put instance files in this directory, or create symbolic links.

The default instance directory depends on Tarantool's `WITH_SYSVINIT` build option: when ON, it is `/etc/tarantool/instances.enabled`, otherwise (OFF or not set) it is `/etc/tarantool/instances.available`. The latter case is typical for Tarantool builds for Linux distros with `systemd`.

To check the build options, say `tarantool --version`.

### Log rotation in tarantooctl

With `tarantoolctl`, *log rotation* is pre-configured to use `logrotate` program, which you must have installed.

File `/etc/logrotate.d/tarantool` is part of the standard Tarantool distribution, and you can modify it to change the default behavior. This is what this file is usually like:

```
/var/log/tarantool/*.log {
    daily
    size 512k
    missingok
    rotate 10
    compress
    delaycompress
    create 0640 tarantool adm
    postrotate
        /usr/bin/tt logrotate `basename ${1%%.*}`
    endscript
}
```

If you use a different log rotation program, you can invoke `tarantoolctl logrotate` command to request instances to reopen their log files after they were moved by the program of your choice.

### Using tarantooctl on FreeBSD

To make `tarantoolctl` work along with `init.d` utilities on FreeBSD, use paths other than those suggested in *Instance configuration*. Instead of `/usr/share/tarantool/` directory, use `/usr/local/etc/tarantool/` and create the following subdirectories:

- `default` for `tarantoolctl` defaults (see example below),
- `instances.available` for all available instance files, and
- `instances.enabled` for instance files to be auto-started by sysvinit.

Here is an example of `tarantoolctl` defaults on FreeBSD:

```
default_cfg = {
    pid_file   = "/var/run/tarantool", -- /var/run/tarantool/${INSTANCE}.pid
    wal_dir    = "/var/db/tarantool", -- /var/db/tarantool/${INSTANCE}/
    snap_dir   = "/var/db/tarantool", -- /var/db/tarantool/${INSTANCE}
    vinyl_dir = "/var/db/tarantool", -- /var/db/tarantool/${INSTANCE}
    logger     = "/var/log/tarantool", -- /var/log/tarantool/${INSTANCE}.log
    username   = "admin"
}

-- instances.available - all available instances
-- instances.enabled - instances to autostart by sysvinit
instance_dir = "/usr/local/etc/tarantool/instances.available"
```

### Starting and stopping instances

While a Lua application is executed by Tarantool, an instance file is executed by `tarantoolctl` which is a Tarantool script.

Here is what `tarantoolctl` does when you issue the command:

```
$ tarantoolctl start <instance_name>
```

1. Read and parse the command line arguments. The last argument, in our case, contains an instance name.
2. Read and parse its own configuration file. This file contains `tarantoolctl` defaults, like the path to the directory where instances should be searched for.

   When `tarantool` is invoked by root, it looks for a configuration file in `/etc/default/tarantool`. When `tarantool` is invoked by a local (non-root) user, it looks for a configuration file first in the current directory (`$PWD/.tarantoolctl`), and then in the current user's home directory (`$HOME/.config/tarantool/tarantool`). If no configuration file is found there, or in the `/usr/local/etc/default/tarantool` file, then `tarantoolctl` falls back to *built-in defaults*.
3. Look up the instance file in the instance directory, for example `/etc/tarantool/instances.enabled`. To build the instance file path, `tarantoolctl` takes the instance name, prepends the instance directory and appends «.lua» extension to the instance file.
4. Override *box.cfg{}* function to pre-process its parameters and ensure that instance paths are pointing to the paths defined in the `tarantoolctl` configuration file. For example, if the configuration file specifies that instance work directory must be in `/var/tarantool`, then the new implementation of `box.cfg{}` ensures that *work_dir* parameter in `box.cfg{}` is set to `/var/tarantool/<instance_name>`, regardless of what the path is set to in the instance file itself.
5. Create a so-called «instance control file». This is a Unix socket with Lua console attached to it. This file is used later by `tarantoolctl` to query the instance state, send commands to the instance and so on.
6. Set the TARANTOOLCTL environment variable to „true“. This allows the user to know that the instance was started by `tarantoolctl`.
7. Finally, use Lua `dofile` command to execute the instance file.

To check the instance file for syntax errors prior to starting `my_app` instance, say:

```
$ tarantoolctl check my_app
```

To stop a running `my_app` instance, say:

```
$ tarantoolctl stop my_app
```

To restart (i.e. stop and start) a running `my_app` instance, say:

```
$ tarantoolctl restart my_app
```

### Running Tarantool locally

Sometimes you may need to run a Tarantool instance locally, e.g. for test purposes. Let's configure a local instance, then start and monitor it with `tarantoolctl`.

First, we create a sandbox directory on the user's path:

```
$ mkdir ~/tarantool_test
```

... and set default `tarantoolctl` configuration in `$HOME/.config/tarantool/tarantool`. Let the file contents be:

```
default_cfg = {
    pid_file  = "/home/user/tarantool_test/my_app.pid",
    wal_dir   = "/home/user/tarantool_test",
    snap_dir  = "/home/user/tarantool_test",
    vinyl_dir = "/home/user/tarantool_test",
    log       = "/home/user/tarantool_test/log",
}
instance_dir = "/home/user/tarantool_test"
```

---

**Примечание:**

- Specify a full path to the user's home directory instead of «~/».
- Omit `username` parameter. `tarantoolctl` normally doesn't have permissions to switch current user when invoked by a local user. The instance will be running under „admin".

---

Next, we create the instance file `~/tarantool_test/my_app.lua`. Let the file contents be:

```
box.cfg{listen = 3301}
box.schema.user.passwd('Gx5!')
box.schema.user.grant('guest','read,write,execute','universe')
fiber = require('fiber')
box.schema.space.create('tester')
box.space.tester:create_index('primary',{})
i = 0
while 0 == 0 do
    fiber.sleep(5)
    i = i + 1
    print('insert ' .. i)
    box.space.tester:insert{i, 'my_app tuple'}
end
```

Let's verify our instance file by starting it without `tarantoolctl` first:

```
$ cd ~/tarantool_test
$ tarantool my_app.lua
2017-04-06 10:42:15.762 [54085] main/101/my_app.lua C> version 1.7.3-489-gd86e36d5b
2017-04-06 10:42:15.763 [54085] main/101/my_app.lua C> log level 5
2017-04-06 10:42:15.764 [54085] main/101/my_app.lua I> mapping 268435456 bytes for tuple␣
↪arena...
2017-04-06 10:42:15.774 [54085] iproto/101/main I> binary: bound to [::]:3301
2017-04-06 10:42:15.774 [54085] main/101/my_app.lua I> initializing an empty data␣
↪directory
```

(continues on next page)

(продолжение с предыдущей страницы)

```
2017-04-06 10:42:15.789 [54085] snapshot/101/main I> saving snapshot `./
↪00000000000000000000.snap.inprogress'
2017-04-06 10:42:15.790 [54085] snapshot/101/main I> done
2017-04-06 10:42:15.791 [54085] main/101/my_app.lua I> vinyl checkpoint done
2017-04-06 10:42:15.791 [54085] main/101/my_app.lua I> ready to accept requests
insert 1
insert 2
insert 3
<...>
```

Now we tell `tarantoolctl` to start the Tarantool instance:

```
$ tarantoolctl start my_app
```

Expect to see messages indicating that the instance has started. Then:

```
$ ls -l ~/tarantool_test/my_app
```

Expect to see the .snap file and the .xlog file. Then:

```
$ less ~/tarantool_test/log/my_app.log
```

Expect to see the contents of `my_app`'s log, including error messages, if any. Then:

```
$ tarantoolctl enter my_app
tarantool> box.cfg{}
tarantool> console = require('console')
tarantool> console.connect('localhost:3301')
tarantool> box.space.tester:select({0}, {iterator = 'GE'})
```

Expect to see several tuples that `my_app` has created.

Stop now. A polite way to stop `my_app` is with `tarantoolctl`, thus we say:

```
$ tarantoolctl stop my_app
```

Finally, we make a cleanup.

```
$ rm -R tarantool_test
```

### Migration from tarantoolctl to tt

*tt* is a command-line utility for managing Tarantool applications that comes to replace `tarantoolctl`. Starting from version 3.0, `tarantoolctl` is no longer shipped as a part of Tarantool distribution; `tt` is the only recommended tool for managing Tarantool applications from the command line.

`tarantoolctl` remains fully compatible with Tarantool 2.* versions. However, it doesn't receive major updates anymore.

We recommend that you migrate from `tarantoolctl` to `tt` to ensure the full support and timely updates and fixes.

### System-wide configuration

`tt` supports system-wide environment configuration by default. If you have Tarantool instances managed by `tarantoolctl` in such an environment, you can switch to `tt` without additional migration steps or use `tt` along with `tarantoolctl`.

Example:

```
$ sudo tt instances
List of enabled applications:
• example

$ tarantoolctl start example
Starting instance example...
Forwarding to 'systemctl start tarantool@example'

$ tarantoolctl status example
Forwarding to 'systemctl status tarantool@example'
 tarantool@example.service - Tarantool Database Server
    Loaded: loaded (/lib/systemd/system/tarantool@.service; enabled; vendor preset:␣
↪enabled)
    Active: active (running)
    Docs: man:tarantool(1)
    Main PID: 6698 (tarantool)
. . .

$ sudo tt status
• example: RUNNING. PID: 6698.

$ sudo tt connect example
• Connecting to the instance...
• Connected to /var/run/tarantool/example.control

/var/run/tarantool/example.control>

$ sudo tt stop example
• The Instance example (PID = 6698) has been terminated.

$ tarantoolctl status example
Forwarding to 'systemctl status tarantool@example'
 tarantool@example.service - Tarantool Database Server
    Loaded: loaded (/lib/systemd/system/tarantool@.service; enabled; vendor preset:␣
↪enabled)
    Active: inactive (dead)
```

### Local configuration

If you have a local `tarantoolctl` configuration, create a `tt` environment based on the existing `.tarantoolctl` configuration file. To do this, run `tt init` in the directory where the file is located.

Example:

```
$ cat .tarantoolctl
default_cfg = {
    pid_file  = "./run/tarantool",
    wal_dir   = "./lib/tarantool",
    memtx_dir = "./lib/tarantool",
    vinyl_dir = "./lib/tarantool",
    log       = "./log/tarantool",
    language  = "Lua",
}
instance_dir = "./instances.enabled"

$ tt init
• Found existing config '.tarantoolctl'
• Environment config is written to 'tt.yaml'
```

After that, you can start managing Tarantool instances in this environment with `tt`:

```
$ tt start app1
• Starting an instance [app1]...

$ tt status app1
• app1: RUNNING. PID: 33837.

$ tt stop app1
• The Instance app1 (PID = 33837) has been terminated.

$ tt check app1
• Result of check: syntax of file '/home/user/instances.enabled/app1.lua' is OK
```

### Commands difference

Most `tarantoolctl` commands look the same in `tt`: `tarantoolctl start` and `tt start`, `tarantoolctl play` and `tt play`, and so on. To migrate such calls, it is usually enough to replace the utility name. There can be slight differences in command flags and format. For details on `tt` commands, see the *tt commands reference*.

The following commands are different in `tt`:

| `tarantoolctl` command | `tt` command |
|---|---|
| `tarantoolctl enter` | `tt connect` |
| `tarantoolctl eval` | `tt connect` with `-f` flag |

**Примечание:** `tt connect` also covers `tarantoolctl connect` with the same syntax.

Example:

```
 # tarantoolctl enter > tt connect
 $ tarantoolctl enter app1
 connected to unix/:./run/tarantool/app1.control
 unix/:./run/tarantool/app1.control>

 $ tt connect app1
 • Connecting to the instance...
 • Connected to /home/user/run/tarantool/app1/app1.control

 # tarantoolctl eval > tt connect -f
 $ tarantoolctl eval app1 eval.lua
 connected to unix/:./run/tarantool/app1.control
 ---
 - 42
 ...

$ tt connect app1 -f eval.lua
 ---
 - 42
 ...

 # tarantoolctl connect > tt connect
 $ tarantoolctl connect localhost:3301
 connected to localhost:3301
 localhost:3301>

 $ tt connect localhost:3301
 • Connecting to the instance...
 • Connected to localhost:3301
```

### 9.3.5 Профилировщик памяти LuaJIT

В Tarantool, начиная с версии *2.7.1*, есть встроенный модуль `misc.memprof`, реализующий профилировщик памяти LuaJIT (далее в разделе — *профилировщик*). Профилировщик предоставляет отчет об аллокации памяти, который помогает проанализировать Lua-код и выявить области наибольшей нагрузки на сборщик мусора на Lua.


- *Использование профилировщика*
  - *Сбор бинарного профиля*
  - *Парсинг бинарного профиля и формирование отчета о профилировании*
- *Вопросы и ответы*
- *Пример анализа отчета о профилировании*
- *Сводка изменений в динамической памяти и параметр –leak-only*

### Использование профилировщика

Работа с профилировщиком состоит из двух этапов:

1. События аллокации, реаллокации и деаллокации в Lua *собираются* в бинарный профиль (далее по тексту — *бинарный профиль памяти* или просто *бинарный профиль*).
2. Выполняется *парсинг* собранного бинарного профиля. В результате формируется отчет о профилировании в удобном для чтения формате.

### Сбор бинарного профиля

Чтобы сформировать бинарный профиль для определенного участка кода на Lua, вставьте этот участок кода между функциями `misc.memprof.start()` и `misc.memprof.stop()`, а затем выполните его в Tarantool.

Below is a chunk of Lua code named `test.lua` to illustrate this.

```
1  -- Отключение аллокации на трассах
2  jit.off()
3  local str, err = misc.memprof.start("memprof_new.bin")
4  -- Lua не создает стековый фрейм для вызова string.rep,
5  -- поэтому все события аллокации
6  -- приписываются не функции append(), а родительской области
7  local function append(str, rep)
8      return string.rep(str, rep)
9  end
10
11 local t = {}
12 for i = 1, 1e4 do
13     -- table.insert - встроенная функция,
14     -- так что профилировщик включает все соответствующие ей
15     -- события аллокации в отчет по основной части кода
16     table.insert(t,
17         append('q', i)
18     )
19 end
20 local stp, err = misc.memprof.stop()
```

В примере test.lua код для запуска профилировщика находится на строке 3:

```
local str, err = misc.memprof.start(ИМЯ_ФАЙЛА)
```

`ИМЯ_ФАЙЛА` — имя бинарного файла, куда записываются профилируемые события.

Если операция завершается ошибкой, функция `misc.memprof.start()` возвращает результат, состоящий из трех частей: `nil`; строка с сообщением об ошибке; код ошибки в зависимости от системы. Ошибка может возникнуть, например, когда не открывается файл для записи или профилировщик уже запущен. Если операция выполняется успешно, `misc.memprof.start()` возвращает `true`.

В примере test.lua код для остановки профилировщика находится на строке 18:

```
local stp, err = misc.memprof.stop()
```

Если операция завершается ошибкой, функция `misc.memprof.stop()` возвращает результат, состоящий из трех частей: `nil`; строка с сообщением об ошибке; код ошибки в зависимости от системы. Ошибка

может возникнуть, например, когда закрыт файловый дескриптор или во время составления отчета происходит сбой. Если операция выполняется успешно, `misc.memprof.stop()` возвращает `true`.

Теперь нужно сгенерировать файл с профилем памяти в бинарном формате. В *примере test.lua* имя файла — `memprof_new.bin`. Выполните код в Tarantool:

```
$ tarantool test.lua
```

Tarantool собирает события аллокации в файл `memprof_new.bin`, помещает его в *рабочую директорию* и завершает сеанс.

*Пример test.lua* также иллюстрирует логику, по которой в некоторых случаях происходит аллокация памяти. Важно понимать эту логику, чтобы *читать* и *анализировать* отчеты о профилировании.

- Строка 2: Рекомендуется отключать JIT-компиляцию, вызывая `jit.off()` перед запуском профилировщика. За дополнительными сведениями обратитесь к *примечанию о jitoff*.
- Строки 6–8: Оптимизация хвостового вызова не создает новый фрейм стека вызовов, поэтому все события аллокации в байт-коде `CALLT/CALLMT` относятся на счет кода, вызывающего эту функцию. Обратите внимание на комментарии перед этими строками.
- Строки 14–16: Как правило, разработчиков несильно интересует информация об аллокации во встроенных функциях LuaJIT. Поэтому если встроенная функция вызывается из функции на Lua, профилировщик собирает все события аллокации для функции на Lua. В противном случае событие относится к функции на C. Обратите внимание на комментарии перед этими строками.

### Парсинг бинарного профиля и формирование отчета о профилировании

After getting the memory profile in binary format, the next step is to parse it to get a human-readable profiling report. You can do this via Tarantool by using the following command (mind the hyphen `-` before the filename):

```
$ tarantool -e 'require("memprof")(arg)' - memprof_new.bin
```

Здесь `memprof_new.bin` — бинарный профиль, *ранее сгенерированный* командой `tarantool test.lua`. (Внимание: в версии Tarantool 2.8.1 поведение команды `tarantool -e` немного изменилось.)

Tarantool генерирует отчет о профилировании и выводит его в консоль перед закрытием сеанса:

```
ALLOCATIONS
@test.lua:14: 10000 events  +50240518 bytes -0 bytes
@test.lua:9: 1 events       +32 bytes       -0 bytes
@test.lua:8: 1 events       +20 bytes       -0 bytes
@test.lua:13: 1 events      +24 bytes       -0 bytes

REALLOCATIONS
@test.lua:13: 13 events     +262216 bytes   -131160 bytes
    Overrides:
        @test.lua:13

@test.lua:14: 11 events     +49536 bytes    -24768 bytes
            Overrides:
        @test.lua:14
        INTERNAL

INTERNAL: 3 events          +8448 bytes     -16896 bytes
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    Overrides:
        @test.lua:14

DEALLOCATIONS
INTERNAL: 1723 events       +0 bytes        -483515 bytes
@test.lua:14: 1 events      +0 bytes        -32768 bytes

HEAP SUMMARY:
@test.lua:14 holds 50248326 bytes: 10010 allocs, 10 frees
@test.lua:13 holds 131080 bytes: 14 allocs, 13 frees
INTERNAL holds 8448 bytes: 3 allocs, 3 frees
@test.lua:9 holds 32 bytes: 1 allocs, 0 frees
@test.lua:8 holds 20 bytes: 1 allocs, 0 frees
```

**Примечание:** On macOS, a report will be different for the same chunk of code because Tarantool and LuaJIT are built with the GC64 mode enabled for macOS.

Рассмотрим структуру отчета. Он состоит из четырех разделов:

- ALLOCATIONS (аллокация)
- REALLOCATIONS (реаллокация)
- DEALLOCATIONS (деаллокация)
- HEAP SUMMARY (сводка изменений в динамической памяти, раздел описан *ниже*)

Записи в каждом разделе отсортированы в порядке от наиболее до наименее частых событий.

Запись о событии регистрируется в следующем формате:

```
@<имя_файла>:<номер_строки>: <количество_событий> events +<выделенные_байты> bytes -
↪<освобожденные_байты> bytes
```

- <имя_файла> — имя файла, содержащего код на Lua.
- <номер_строки> — номер строки, в которой обнаружено событие.
- <количество_событий> — количество событий для этой строки кода.
- +<выделенная_память_в_байтах> bytes — объем выделенной памяти для всех событий в этой строке.
- +<освобожденная_память_в_байтах> bytes — объем освобожденной памяти для всех событий в этой строке.

Метка `Overrides` на событии показывает, какое событие аллокации оно замещает.

См. *код примера test.lua выше* и пояснения в комментариях к нему.

Метка `INTERNAL` показывает, что событие произошло во внутренних структурах LuaJIT.

**Примечание:** Важное примечание о метке `INTERNAL` и рекомендации отключать компиляцию JIT (`jit.off()`). В текущей версии профилировщика не поддерживается подробное (verbose) описание событий аллокации при трассировке. Если память выделяется во время трассировки, профилировщик не может соотнести события аллокации с соответствующей частью кода. В этом случае профилировщик помечает такие события как `INTERNAL`.

Когда компиляция JIT включена, выполняется несколько процессов трассировки. В результате пометку `INTERNAL` в отчете о профилировании получают самые разные события: некоторые из них действительно возникают во внутренних структурах LuaJIT, а другие вызваны аллокацией при трассировке.

Вы можете получить более конкретный отчет, в который не входят события аллокации, вызванные JIT-компилятором. Для этого до запуска профилировщика *вызовите jit.off()*. Чтобы полностью исключить из отчета аллокацию на трассах, удалите предыдущие трассы, вызвав после `jit.off()` функцию `jit.flush()`.

Отключать компиляцию JIT перед запуском профилировщика рекомендуется, но это не обязательно. Например, в производственной среде без компиляции JIT полное представление об аллокациях памяти получить невозможно. В подобных случаях большая часть событий `INTERNAL`, вероятнее всего, происходит при трассировках.

---

Иногда отчет о профилировании помогает исследовать код на Lua. Однако этот метод подходит не для любого кода, так что рекомендаций на этот счет нет. Посмотрите *пример анализа отчета о профилировании*, чтобы узнать, как можно использовать отчет для исследования кода.

Больше информации о том, как использовать профилировщик, вы найдете ниже в разделе *Вопросы и ответы*.

### Вопросы и ответы

В этом разделе даются ответы на часто задаваемые вопросы о работе профилировщика.

**Вопрос (B)**: Проверяет ли профилировщик аллокацию в C и коде на C?

**Ответ (A)**: Профилировщик включает в отчет только события выделения памяти Lua-аллокатором. Отчет содержит все события аллокации в Lua, например создание таблиц и строк. Однако профилировщик не отслеживает выделение памяти с помощью `malloc()` или действия аллокаторов, не связанных с Lua. Чтобы фиксировать такие события, для отладки можно использовать `valgrind`.

**В**: Почему у меня в отчете о профилировании столько событий аллокации `INTERNAL`? Что это значит?

**О**: Пометка `INTERNAL` означает, что событие аллокации/реаллокации/деаллокации связано с внутренними структурами LuaJIT или трассами. Сейчас профилировщик не включает в отчет подробное описание событий аллокации для объектов, создаваемых на трассах. Попробуйте перед запуском профилировщика *добавить jit.off()*.

**В**: Почему на некоторых событиях аллокации/деаллокации нет метки `Overrides`?

**О**: Вероятно, события связаны с объектами, созданными до запуска профилировщика. Если перед запуском профилировщика добавить вызов `collectgarbage()`, то будут учитываться и события, связанные с «мертвыми» объектами: под такие объекты ранее была выделена память, но на момент запуска профилировщика они уже недоступны.

**В**: Почему некоторые объекты не учитываются при профилировании? Это связано с утечкой памяти?

**О**: LuaJIT использует инкрементальный сборщик мусора. Когда профилировщик завершает работу, цикл сборки мусора может все еще выполняться. Чтобы обеспечить профилирование «мертвых» объектов, добавьте `collectgarbage()` перед командой остановки профилировщика.

**В**: Можно ли профилировать не только часть кода, а все приложение? Можно ли запустить профилировщик во время работы приложения?

**О**: Да. Вот пример кода, который можно вставить в консоль Tarantool, когда приложение уже запущено:

```
local fiber = require "fiber"
local log = require "log"

fiber.create(function()
  fiber.name("memprof")

  collectgarbage() -- Сбор мертвых объектов
  log.warn("start of profile")

  local st, err = misc.memprof.start(ИМЯ_ФАЙЛА)
  if not st then
    log.error("failed to start profiler: %s", err)
  end

  fiber.sleep(ВРЕМЯ)

  collectgarbage()
  st, err = misc.memprof.stop()

  if not st then
    log.error("profiler on stop error: %s", err)
  end

  log.warn("end of profile")
end)
```

Пояснения:

- `ИМЯ_ФАЙЛА` — имя бинарного файла, куда записываются профилируемые события.
- `ВРЕМЯ` — продолжительность профилирования в секундах.

Кроме того, вызывать `misc.memprof.start()` и `misc.memprof.stop()` можно напрямую из консоли.

### Пример анализа отчета о профилировании

Ниже приведен код на Lua — файл `format_concat.lua`. Этот код будет исследован с применением отчетов о профилировании памяти.

```
-- Отключение аллокации на новых трассах
jit.off()

local function concat(a)
  local nstr = a.."a"
  return nstr
end

local function format(a)
  local nstr = string.format("%sa", a)
  return nstr
end

collectgarbage()

local binfile = "/tmp/memprof_"..(arg[0]):match("([^/]*).lua")..".bin"

local st, err = misc.memprof.start(binfile)
assert(st, err)

-- Нагрузка
for i = 1, 10000 do
  local f = format(i)
  local c = concat(i)
end
collectgarbage()

local st, err = misc.memprof.stop()
assert(st, err)

os.exit()
```

Запустив этот код *в Tarantool* и выполнив последующий *парсинг* бинарного профиля в /tmp/memprof_format_concat.bin, вы получите такой отчет о профилировании:

```
ALLOCATIONS
@format_concat.lua:10: 19996 events +624284 bytes   -0 bytes
INTERNAL: 1 events                  +65536 bytes    -0 bytes

REALLOCATIONS

DEALLOCATIONS
INTERNAL: 19996 events              +0 bytes        -558778 bytes
    Overrides:
        @format_concat.lua:10

@format_concat.lua:10: 2 events     +0 bytes        -98304 bytes
    Overrides:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        @format_concat.lua:10

HEAP SUMMARY:
INTERNAL holds 65536 bytes: 1 allocs, 0 frees
```

Отчет может вызвать такие вопросы:

- Почему нет событий по функции `concat()`?
- Почему количество событий — это не целое число?
- Why are there about 20K allocations instead of 10K?

Во-первых, LuaJIT не создает новую строку, если уже есть строка с такой же нагрузкой (подробнее на сайте lua-users.org/wiki). Это называется интернированием строк. Иными словами, если строка создана с помощью функции `format()`, нет необходимости создавать такую же строку с помощью функции `concat()` — LuaJIT будет использовать предыдущую.

По этой же причине количество событий аллокации — это не целое число, как можно было бы ожидать при использовании оператора цикла `for i = 1, 10000...`. Некоторые строки Tarantool создает для внутренних нужд и встроенных модулей, поэтому часть строк уже существует.

Но откуда столько событий аллокации? Их почти в 2 раза больше, чем можно было бы ожидать. Это происходит потому, что встроенная функция `string.format()` каждый раз создает дополнительную строку для идентификатора `%s`. То есть в каждой итерации регистрируются два события аллокации: для `tostring(i)` и для `string.format("%sa", string_i_value)`. Добавив строку `local _ = tostring(i)` между строками 22 и 23, вы увидите разницу в поведении.

To profile only the `concat()` function, comment out line 23 (which is `local f = format(i)`) and run the profiler. Now the output looks like this:

```
ALLOCATIONS
@format_concat.lua:5: 10000 events  +284411 bytes    -0 bytes

REALLOCATIONS

DEALLOCATIONS
INTERNAL: 10000 events              +0 bytes         -218905 bytes
    Overrides:
        @format_concat.lua:5

@format_concat.lua:5: 1 events      +0 bytes         -32768 bytes

HEAP SUMMARY:
@format_concat.lua:5 holds 65536 bytes: 10000 allocs, 9999 frees
```

**В**: Что изменится, если включить компиляцию JIT?

**О**: Закомментируйте вторую строку (`jit.off()`) в *коде* и запустите профилировщик. Теперь в отчете только 56 событий. Остальные события связаны с JIT (см. также соответствующую задачу на GitHub):

```
ALLOCATIONS
@format_concat.lua:5: 56 events +1112 bytes -0 bytes
@format_concat.lua:0: 4 events  +640 bytes  -0 bytes
INTERNAL: 2 events              +382 bytes  -0 bytes
```

(continues on next page)

(продолжение с предыдущей страницы)

```
REALLOCATIONS

DEALLOCATIONS
INTERNAL: 58 events             +0 bytes    -1164 bytes
    Overrides:
        @format_concat.lua:5
        INTERNAL


HEAP SUMMARY:
@format_concat.lua:0 holds 640 bytes: 4 allocs, 0 frees
INTERNAL holds 360 bytes: 2 allocs, 1 frees
```

Так произошло потому, что трассировка была скомпилирована после 56 итераций (это значение параметра компилятора `hotloop` по умолчанию). Затем JIT-компилятор удалил из трассировки неиспользуемую переменную `c`, а вместе с ней и неиспользуемый код функции `concat()`.

Теперь включите компиляцию JIT и запустите профилирование только для функции `format()`. Для этого закомментируйте строки 2 и 24 (`jit.off()` и `local c = concat(i)`), раскомментируйте строку 23 (`local f = format(i)`) и вызовите профилировщик. Результат будет такой:

```
ALLOCATIONS
@format_concat.lua:10: 19996 events +624284 bytes  -0 bytes
INTERNAL: 4 events                  +66928 bytes   -0 bytes
@format_concat.lua:0: 4 events      +640 bytes     -0 bytes

REALLOCATIONS

DEALLOCATIONS
INTERNAL: 19997 events              +0 bytes       -559034 bytes
    Overrides:
        @format_concat.lua:0
        @format_concat.lua:10

@format_concat.lua:10: 2 events     +0 bytes       -98304 bytes
    Overrides:
        @format_concat.lua:10


HEAP SUMMARY:
INTERNAL holds 66928 bytes: 4 allocs, 0 frees
@format_concat.lua:0 holds 384 bytes: 4 allocs, 1 frees
```

**В**: Откуда так много событий аллокации по сравнению с `concat()`?

**О**: Ответ прост: LuaJIT еще не скомпилировал функцию `string.format()` с идентификатором `%s`. Поэтому трассировка не регистрируется, а компилятор не выполняет соответствующую оптимизацию.

Изменим функцию `format` в строках 9–12 *примера для анализа отчета о профилировании* следующим образом:

```
local function format(a)
  local nstr = string.format("%sa", tostring(a))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  return nstr
end
```

Теперь отчет о профилировании будет выглядеть намного лучше:

```
ALLOCATIONS
@format_concat.lua:10: 109 events   +2112 bytes -0 bytes
@format_concat.lua:0: 4 events      +640 bytes  -0 bytes
INTERNAL: 3 events                  +1206 bytes -0 bytes

REALLOCATIONS

DEALLOCATIONS
INTERNAL: 112 events                +0 bytes    -2460 bytes
    Overrides:
        @format_concat.lua:0
        @format_concat.lua:10
        INTERNAL


HEAP SUMMARY:
INTERNAL holds 1144 bytes: 3 allocs, 1 frees
@format_concat.lua:0 holds 384 bytes: 4 allocs, 1 frees
```

#### Сводка изменений в динамической памяти и параметр –leak-only

Эта функциональная возможность появилась в версии *2.8.1*.

В конце каждого отчета приводится раздел HEAP SUMMARY (сводка изменений в динамической памяти). Этот раздел выглядит так:

```
@<имя_файла>:<номер_строки> holds <количество_доступных_байтов> bytes: <количество_
↪событий_аллокации> allocs, <количество_событий_деаллокации> frees
```

Иногда программа может вызывать множество событий деаллокации. В этом случае раздел DEALLOCATION сильно увеличится и отчет будет сложно читать. Чтобы уменьшить количество выводимых данных, запустите парсинг с дополнительным параметром `--leak-only`. Например, так:

```
$ tarantool -e 'require("memprof")(arg)' - --leak-only memprof_new.bin
```

При использовании параметра `--leak-only` выводится только раздел HEAP SUMMARY.

### 9.3.6 Метрики LuaJIT

Tarantool может возвращать метрики текущего экземпляра через Lua API или C API.

- *misc.getmetrics()*
- *Значения в таблице getmetrics*
- *C API для getmetrics*

- *Пример с gc_strnum, strhash_miss и strhash_hit*
- *Пример с gc_allocated и gc_freed*
- *Пример с gc_allocated и оптимизацией по памяти*
- *gc_steps_atomic и gc_steps_propagate*
- *Пример с jit_trace_num и jit_trace_abort*
- *Пример с jit_snap_restore и деоптимизацией производительности*

### misc.getmetrics()

getmetrics()

Получение значений метрик в виде таблицы.

Параметры: нет

**возвращает** таблицу

Пример: `metrics_table = misc.getmetrics()`

### Значения в таблице getmetrics

Таблица метрик содержит 19 значений типа `number`. Они получены в результате приведения к вещественному типу двойной точности (double), что позволяет практически не терять в точности исходных значений. Значения, имена которых начинаются на `gc_`, связаны со сборщиком мусора в LuaJIT. Полную информацию о сборщике мусора можно найти на вики-странице lua-users и слайде от создателя языка Lua. Значения, имена которых начинаются на `jit_`, связаны с фазами JIT-компиляции. Более подробно фазы описаны в научной работе, написанной в рамках одного из проектов ЦЕРН.

Монотонными называются суммарные значения, которые подсчитываются с момента начала работы, а не с момента последнего вызова getmetrics(). Возможно переполнение значений.

Поскольку многие значения монотонны, анализ обычно проходит так: вызывается `getmetrics()`, полученная таблица сохраняется, а затем `getmetrics()` вызывается вновь и новая таблица сравнивается с сохраненной. Разницу между значениями, измеренными в два момента времени, называют кривой наклона. Интерес может представлять, например, кривая наклона, которая демонстрирует ускорение. Разница между последним и предыдущим значениями на такой кривой все время увеличивается. Для некоторых элементов таблицы ниже будут приведены примеры.

| Имя | Содержимое | Монотонное? |
|---|---|---|
| gc_allocated | количество выделенной памяти в байтах | да |
| gc_cdatanum | количество размещенных объектов cdata | нет |
| gc_freed | количество освобожденной памяти в байтах | да |
| gc_steps_atomic | количество шагов сборщика мусора, фаза atomic, инкрементальная | да |
| gc_steps_finalize | количество шагов сборщика мусора, фаза finalize | да |
| gc_steps_pause | количество шагов сборщика мусора, фаза pauses | да |
| gc_steps_propagate | количество шагов сборщика мусора, фаза propagate | да |
| gc_steps_sweep | number of steps of garbage collector, sweep phases (see the Sweep phase description) | да |
| gc_steps_sweepstring | количество шагов сборщика мусора, фаза sweep для строк | да |
| gc_strnum | количество размещенных объектов-строк | нет |
| gc_tabnum | количество размещенных объектов-таблиц | нет |
| gc_total | текущее количество выделенной памяти в байтах (обычно равно разности gc_allocated и gc_freed) | нет |
| gc_udatanum | количество размещенных объектов udata | нет |
| jit_mcode_size | общий объем выделенного машинного кода | нет |
| jit_snap_restore | общее количество восстановлений стека по снимку (сработавших защитных утверждений, которые привели к остановке выполнения трассы). См. внешнее руководство по SNAP. | да |
| jit_trace_abort | общее количество прерванных трассировок | да |
| jit_trace_num | количество JIT-трассировок | нет |
| strhash_hit | количество интернированных строк (если строка уже есть в пуле, новая копия не создается и память под нее не выделяется) | да |
| strhash_miss | общее количество памяти, выделенной для строк за время жизни платформы | да |

Примечание. Функция ujit.getmetrics() возвращает похожие имена. Однако многие значения, используемые в uJIT, не монотонны.

Note: Although value names are similar to value names in *LuaJIT metrics*, and the values are exactly the same, misc.getmetrics() is slightly easier because there is no need to 'require' the misc module.

### C API для getmetrics

Lua-функция `getmetrics()` — обертка для C-функции `luaM_metrics()`.

Программы на C могут включать заголовок `libmisclib.h`, куда входят следующие определения:

```
struct luam_Metrics { /* имена, описанные ранее для Lua */ }

LUAMISC_API void luaM_metrics(lua_State *L, struct luam_Metrics *metrics);
```

Имена элементов структуры `luam_Metrics` совпадают с именами в *таблице значений getmetrics* для Lua. У всех элементов структуры `luam_Metrics` тип данных — `size_t`. Функция `luaM_metrics()` заполняет структуру `*metrics` метриками, относящимися к Lua-состоянию, которое связано с корутиной `L`.

#### Пример с программой на языке C

В руководстве Tarantool по *хранимым процедурам на языке C* перейдите к примеру с файлом `easy.c`. Удалите содержимое файла и вставьте следующий код:

```
#include "module.h"
#include <lmisclib.h>

int easy(box_function_ctx_t *ctx, const char *args, const char *args_end)
{
  lua_State *ls = luaT_state();
  struct luam_Metrics m;
  luaM_metrics(ls, &m);
  printf("allocated memory = %lu\n", m.gc_allocated);
  return 0;
}
```

Теперь, как в исходном примере, выполните через клиент запросы до `capi_connection:call('easy')` включительно. На экране появится следующее: `allocated memory = 4431950` (число приведено для примера).

### Пример с gc_strnum, strhash_miss и strhash_hit

Отслеживать размещение новых строковых объектов можно так:

```
function f()
  collectgarbage("collect")
  local oldm = misc.getmetrics()
  local table_of_strings = {}
  for i = 3000, 4000 do table.insert(table_of_strings, tostring(i)) end
  for i = 3900, 4100 do table.insert(table_of_strings, tostring(i)) end
  local newm = misc.getmetrics()
  print("gc_strnum diff = " .. newm.gc_strnum - oldm.gc_strnum)
  print("strhash_miss diff = " .. newm.strhash_miss - oldm.strhash_miss)
  print("strhash_hit diff = " .. newm.strhash_hit - oldm.strhash_hit)
end
f()
```

Вероятный результат — `gc_strnum diff = 1101`, так как мы добавили 1202 строки, 101 из которых были дубликатами. По той же причине `strhash_miss = 1101`, а `strhash_hit = 101` плюс некоторые издержки. (`strhash_hit` всегда предполагает небольшие издержки, которые можно игнорировать.)

Результат лишь вероятный, поскольку память для строк могла быть выделена ранее. Хорошо, если кривая наклона `strhash_miss` менее крутая, чем у `strhash_hit`.

Доступ к остальным значениям `gc_*num` — `gc_cdatanum`, `gc_tabnum` и `gc_udatanum` можно получить аналогичным образом. Любое значение `gc_*num` поможет при поиске утечки памяти: общее количество этих объектов не должно постоянно расти. Более общий способ искать утечки памяти — наблюдать за переменной `gc_total`. Также можно отслеживать значение `jit_mcode_size`, отражающее объем памяти, выделенной для трасс машинного кода.

### Пример с gc_allocated и gc_freed

Чем меньше работает сборщик мусора, тем лучше. Отслеживать, насколько его нагружает приложение, можно так:

```
function f()
  for i = 1, 10 do collectgarbage("collect") end
  local oldm = misc.getmetrics()
  local newm = misc.getmetrics()
  oldm = misc.getmetrics()
  collectgarbage("collect")
  newm = misc.getmetrics()
  print("gc_allocated diff = " .. newm.gc_allocated - oldm.gc_allocated)
  print("gc_freed diff = " .. newm.gc_freed - oldm.gc_freed)
end
f()
```

Результат: `gc_allocated diff = 800`, `gc_freed diff = 800`. Отсюда видно, что строка `local ... = getmetrics()` вызывает аллокацию памяти, поскольку создает таблицу и наполняет ее значениями. Когда имя переменной (в данном случае `oldm`) используется повторно, память освобождается. Обычно это происходит не сразу, однако `collectgarbage("collect")` принудительно запускает сборку мусора, благодаря чему можно немедленно увидеть результат.

### Пример с gc_allocated и оптимизацией по памяти

Этот пример показывает, что можно оптимизировать расход памяти для таблиц.

```
function f()
  collectgarbage("collect")
  local oldm = misc.getmetrics()
  local t = {}
  for i = 1, 513 do
    t[i] = i
  end
  local newm = misc.getmetrics()
  local diff = newm.gc_allocated - oldm.gc_allocated
  print("diff = " .. diff)
end
f()
```

Результат покажет, что значение diff примерно равно 18000.

Если инициализировать таблицу по-другому, получится вот что:

```
function f()
  local table_new = require "table.new"
  local oldm = misc.getmetrics()
  local t = table_new(513, 0)
  for i = 1, 513 do
    t[i] = i
  end
  local newm = misc.getmetrics()
  local diff = newm.gc_allocated - oldm.gc_allocated
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  print("diff = " .. diff)
end
f()
```

Результат покажет, что значение diff примерно равно 6000.

### gc_steps_atomic и gc_steps_propagate

Нагрузку на сборщик мусора помогают отслеживать кривые наклона метрик `gc_steps_*`. Во время долго выполняющихся процедур значения `gc_steps_*` увеличиваются. При этом большие промежутки времени между увеличениями `gc_steps_atomic` — хороший признак. Поскольку `gc_steps_atomic` увеличивается только раз в цикл сбора мусора, можно увидеть, сколько циклов прошло.

Кроме того, по тому, насколько выросло значение `gc_steps_propagate`, можно косвенно оценить количество объектов. Это значение также коррелирует с множителем шага сборщика мусора. Например, количество инкрементальных шагов может увеличиваться, однако множитель шага настроен так, что за каждый шаг можно обработать лишь небольшое количество объектов. Поэтому при настройке сборщика мусора следует учитывать эти показатели.

Следующая функция оценивает, вызывает ли оператор SQL большую нагрузку:

```
function f()
  collectgarbage("collect")
  local oldm = misc.getmetrics()
  collectgarbage("collect")
  box.execute([[DROP TABLE _vindex;]])
  local newm = misc.getmetrics()
  print("gc_steps_atomic = " .. newm.gc_steps_atomic - oldm.gc_steps_atomic)
  print("gc_steps_finalize = " .. newm.gc_steps_finalize - oldm.gc_steps_finalize)
  print("gc_steps_pause = " .. newm.gc_steps_pause - oldm.gc_steps_pause)
  print("gc_steps_propagate = " .. newm.gc_steps_propagate - oldm.gc_steps_propagate)
  print("gc_steps_sweep = " .. newm.gc_steps_sweep - oldm.gc_steps_sweep)
end
f()
```

Очевидно, значения метрик `gc_steps_ *`, полученные до вызова `box.execute()`, существенно не отличаются от значений, полученных после этого вызова.

### Пример с jit_trace_num и jit_trace_abort

JIT-компиляторы трассируют код, пытаясь найти возможность для компиляции, чтобы повысить производительность. Значение `jit_trace_abort` показывает, как часто эти попытки оканчиваются неудачей (чем меньше это значение, тем лучше), а `jit_trace_num` — как много трассировок сгенерировано с момента последнего выполнения операции flush (обычно чем больше значение, тем лучше).

Код следующей функции будет успешно трассирован:

```
function f()
  jit.flush()
  for i = 1, 10 do collectgarbage("collect") end
  local oldm = misc.getmetrics()
  collectgarbage("collect")
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  local sum = 0
  for i = 1, 57 do
    sum = sum + 57
  end
  for i = 1, 10 do collectgarbage("collect") end
  local newm = misc.getmetrics()
  print("trace_num = " .. newm.jit_trace_num - oldm.jit_trace_num)
  print("trace_abort = " .. newm.jit_trace_abort - oldm.jit_trace_abort)
end
f()
```

Результат: `trace_num = 1`, `trace_abort = 0`. Отлично.

А в следующей функции, похоже, есть код, способный вызвать у LuaJIT проблемы:

```
jit.opt.start(0, "hotloop=2", "hotexit=2", "minstitch=15")
_G.globalthing = 5
function f()
  jit.flush()
  collectgarbage("collect")
  local oldm = misc.getmetrics()
  collectgarbage("collect")
  local sum = 0
  for i = 1, box.space._vindex:count()+ _G.globalthing do
    box.execute([[SELECT RANDOMBLOB(0);]])
    require('buffer').ibuf()
    _G.globalthing = _G.globalthing - 1
  end
  local newm = misc.getmetrics()
  print("trace_num = " .. newm.jit_trace_num - oldm.jit_trace_num)
  print("trace_abort = " .. newm.jit_trace_abort - oldm.jit_trace_abort)
end
f()
```

Результат: `trace_num` — от 2 до 4, `trace_abort = 1`. Это означает, что нужно было сгенерировать до четырех трассировок вместо одной, причем что-то заставило LuaJIT прекратить попытки. Дальнейшая трассировка показывает, что проблема не в подозрительно выглядящих операторах внутри функции, а в вызове `jit.opt.start`. (Содержимое файла `jit.dump` может помочь разобраться, как протекала компиляция трасс.)

#### Пример с jit_snap_restore и деоптимизацией производительности

Если кривая наклона метрики `jit_snap_restore` после изменений в старом коде растет, это может означать, что LuaJIT чаще останавливает выполнение кода на трассах. А это, в свою очередь, может указывать на снижение производительности.

Рассмотрим следующий код:

```
function f()
  local function foo(i)
    return i <= 5 and i or tostring(i)
  end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  -- параметр minstitch нужен, чтобы эмулировать поведение non-stitching
  jit.opt.start(0, "hotloop=2", "hotexit=2", "minstitch=15")
  local sum = 0
  local oldm = misc.getmetrics()
  for i = 1, 10 do
    sum = sum + foo(i)
  end
  local newm = misc.getmetrics()
  local diff = newm.jit_snap_restore - oldm.jit_snap_restore
  print("diff = " .. diff)
end
f()
```

Результат: `diff = 3`, поскольку при выполнении трассы возможны три сторонних выхода. Один находится в конце цикла. Другие два переключают выполнение на интерпретатор, прежде чем LuaJIT решит, что фрагмент кода «горячий» (значение параметра `hotloop` по умолчанию равно `56` согласно документации LuaJIT).

А теперь изменим единственную строку в функции `local foo`, чтобы получить следующий код:

```
function f()
  local function foo(i)
    -- функция math.fmod еще не скомпилирована
    return i <= 5 and i or math.fmod(i, 11)
  end
  -- параметр minstitch нужен, чтобы эмулировать поведение non-stitching
  jit.opt.start(0, "hotloop=2", "hotexit=2", "minstitch=15")
  local sum = 0
  local oldm = misc.getmetrics()
  for i = 1, 10 do
    sum = sum + foo(i)
  end
  local newm = misc.getmetrics()
  local diff = newm.jit_snap_restore - oldm.jit_snap_restore
  print("diff = " .. diff)
end
f()
```

Результат: значение `diff` увеличилось, так как сторонних выходов стало больше. Этот тест показывает, что измение кода влияет на производительность.

## 9.4 SQL reference

This reference covers all the SQL statements and clauses supported by Tarantool.

### 9.4.1 What Tarantool's SQL product delivers

Tarantool's SQL is a major new feature that was first introduced with Tarantool version 2.1. The primary advantages are: - a high level of SQL compatibility - an easy way to switch from NoSQL to SQL and back - the Tarantool brand.

The «high level of SQL compatibility» includes support for joins, subqueries, triggers, indexes, groupings, transactions in a multi-user environment, and conformance with the majority of the mandatory requirements of the SQL:2016 standard.

The «easy way to switch» consists of the fact that the same tables can be operated on with SQL and with the long-established Tarantool-NoSQL product, meaning that when you want standard Relational-DBMS jobs you can do them, and when you want NoSQL capability you can have it (Tarantool-NoSQL outperforms other NoSQL products in public benchmarks).

The «Tarantool brand» comes from the support of a multi-billion-dollar internet / mail / social-network provider, a dozens-of-professionals staff of programmers and support people, a community who believes in open-source BSD licensing, and hundreds of corporations / government bodies using Tarantool products in production already.

The status of Tarantool's SQL feature is «release». So, it is working now and you can verify that by downloading it and trying all the features, which will be explained in the rest of this document. There is also a *tutorial*.

#### Differences from other products

Differences from other SQL products: The Tarantool design requirement is that Tarantool's SQL conforms to the majority of the listed mandatory requirements of the core SQL:2016 standard, and this will be shown in the specific conformance statements in the feature list in a section about *«compliance with the official SQL standard»*. Possibly the deviations which most people will find notable are: type checking is less strict, and some data definition options must be done with NoSQL syntax.

Differences from other NoSQL products: By examining attempts by others to paste relatively smaller subsets of SQL onto NoSQL products, it should be possible to conclude that Tarantool's SQL has demonstrably more features and capabilities. The reason is that the Tarantool developers started with a complete code base of a working SQL DBMS and made it work with Tarantool-NoSQL underneath, rather than starting with a NoSQL DBMS and adding syntax to it.

#### What Tarantool's SQL manual delivers

The following parts of this document are: The *SQL User Guide* explains «How to get Started» and explains the terms and the syntax elements that apply for all SQL statements. The *SQL Statements and Clauses* guide explains, for each SQL statement, the format and the rules and the exceptions and the examples and the limitations. The *SQL Plus Lua* guide has the details about calling Lua from SQL, calling SQL from Lua, and using the same database objects in both SQL and Lua. The *SQL Features* list shows how the product conforms with the mandatory features of the SQL standard.

Users are expected to know what databases are, and experience with other SQL DBMSs would be an advantage. To learn about the basics of relational database management and SQL in particular, check the *SQL Beginners" Guide* in the *How-to guides* section.

### 9.4.2 SQL user guide

The User Guide describes how users can start up with SQL with Tarantool, and necessary concepts.

| Heading | Введение |
|---|---|
| *Getting Started* | Typing SQL statements on a console |
| *Supported Syntax* | For example what characters are allowed |
| *Concepts* | tokens, literals, identifiers, operands, operators, expressions, statements |
| *Data type conversion* | Casting, implicit or explicit |

#### Getting Started

Пояснения по установке и запуску сервера Tarantool приведены в других главах руководства по Tarantool.

To get started specifically with the SQL features, using Tarantool as a client, execute these requests:

```
box.cfg{}
box.execute([[VALUES ('hello');]])
```

The bottom of the screen should now look like this:

```
tarantool> box.execute([[VALUES ('hello');]])
---
- metadata:
  - name: COLUMN_1
    type: string
  rows:
  - ['hello']
...
```

That's an SQL statement done with Tarantool.

Now you are ready to execute any SQL statements via the connection. For example

```
box.execute([[CREATE TABLE things (id INTEGER PRIMARY key,
                                   remark STRING);]])
box.execute([[INSERT INTO things VALUES (55, 'Hello SQL world!');]])
box.execute([[SELECT * FROM things WHERE id > 0;]])
```

And you will see the results of the SQL query.

For the rest of this chapter, the *box.execute([[. . . ]])* enclosure will not be shown. Examples will simply say what a piece of syntax looks like, such as `SELECT 'hello';` and users should know that must be entered as `box.execute([[SELECT 'hello';]])` It is also legal to enclose SQL statements inside single or double quote marks instead of [[ . . . ]].

### Supported Syntax

Keywords, for example CREATE or INSERT or VALUES, may be entered in either upper case or lower case.

Literal values, for example `55` or `'Hello SQL world!'`, should be entered without single quote marks if they are numeric, and should be entered with single quote marks if they are strings.

Object names, for example table1 or column1, should usually be entered without double quote marks and are subject to some restrictions. They may be enclosed in double quote marks and in that case they are subject to fewer restrictions.

Almost all keywords are *reserved*, which means that they cannot be used as object names unless they are enclosed in double quote marks.

Comments may be between `/*` and `*/` (bracketed) or between `--` and the end of a line (simple).

```
INSERT /* This is a bracketed comment */ INTO t VALUES (5);
INSERT INTO t VALUES (5); -- this is a simple comment
```

Выражения, например `a + b` или `a > b AND NOT a <= b`, могут содержать арифметические операторы `+ - / *` и операторы сравнения `= > < <= >= LIKE`, а также могут использоваться вместе с `AND OR NOT`, при этом круглые скобки необязательны.

### Concepts

In the *SQL beginners“ guide* there was discussion of: What are: relational databases, tables, views, rows, and columns? What are: transactions, write-ahead logs, commits and rollbacks? What are: security considerations? How to: add, delete, or update rows in tables? How to: work inside transactions with commits and/or rollbacks? How to: select, join, filter, group, and sort rows?

Tarantool has a «schema». A schema is a container for all database objects. A schema may be called a «database» in other DBMS implementations

Tarantool allows four types of «database objects» to be created within the schema: tables, triggers, indexes, and constraints. Within tables, there are «columns».

Almost all Tarantool SQL statements begin with a reserved-word «verb» such as INSERT, and end optionally with a semicolon. For example: `INSERT INTO t VALUES (1);`

A Tarantool SQL database and a Tarantool NoSQL database are the same thing. However, some operations are only possible with SQL, and others are only possible with NoSQL. Mixing SQL statements with NoSQL requests is allowed.

### Tokens

The token is the minimum SQL-syntax unit that Tarantool understands. These are the types of tokens:

Ключевые слова — официальные слова языка, например `SELECT`. Литералы — числовые или строковые константы, например `15.7` или `'Taranto'`. Идентификаторы — имена объектов, например `column55` или `table_of_accounts`. Операторы (строго говоря, неалфавитные операторы) — математические операторы, например `* / + - ( ) , ; < = >=`.

Токены могут быть отделены друг от друга одним или несколькими разделителями. * Символы-разделители: табуляция (U+0009), перевод строки (U+000A), вертикальная табуляция (U+000B), смена страницы (U+000C), возврат каретки (U+000D), пробел (U+0020), следующая строка (U+0085), а также все редкие символы классов Zl, Zp и Zs Юникода. Полный список символов вы найдете на странице https://github.com/tarantool/tarantool/issues/2371. * Комментарии (начинаются с `/*`

и заканчиваются */). * Однострочные комментарии (начинаются с -- и заканчиваются переводом строки). Разделители не нужны ни перед операторами, ни после них. Разделители необходимы после ключевых слов, числовых значений или обычных идентификаторов, если только следующий токен не является оператором. Таким образом, Tarantool может понять следующую серию из шести токенов: SELECT'a'FROM/**/t; Но для удобства чтения токены обычно разделяют пробелами: SELECT 'a' FROM /**/ t;

### Literals

There are eight kinds of literals: BOOLEAN INTEGER DOUBLE DECIMAL STRING VARBINARY MAP ARRAY.

BOOLEAN literals: TRUE | FALSE | UNKNOWN A literal has *data type = BOOLEAN* if it is the keyword TRUE or FALSE. UNKNOWN is a synonym for NULL. A literal may have type = BOOLEAN if it is the keyword NULL and there is no context to indicate a different data type.

INTEGER literals: [plus-sign | minus-sign] digit [digit ...] or, for a hexadecimal integer literal, [plus-sign | minus-sign] 0X | 0x hexadecimal-digit [hexadecimal-digit ...] Examples: 5, -5, +5, 55555, 0X55, 0x55 Hexadecimal 0X55 is equal to decimal 85. A literal has *data type = INTEGER* if it contains only digits and is in the range -9223372036854775808 to +18446744073709551615, integers outside that range are illegal.

DOUBLE literals: [E|e [plus-sign | minus-sign] digit ...] Examples: 1E5, 1.1E5. A literal has *data type = DOUBLE* if it contains «E». DOUBLE literals are also known as floating-point literals or approximate-numeric literals. To represent «Inf» (infinity), write a real numeric outside the double-precision numeric range, for example 1E309. To represent «nan» (not a number), write an expression that does not result in a real numeric, for example 0/0, using Tarantool/NoSQL. This will appear as NULL in Tarantool/SQL. In an earlier version literals containing periods were considered to be *NUMBER* literals. In a future version «nan» may not appear as NULL. Prior to Tarantool v. 2.10.0, digits with periods such as .0 were considered to be DOUBLE literals, but now they are considered to be DECIMAL literals.

DECIMAL literals: [plus-sign | minus-sign] [digit [digit ...]] period [digit [digit ...]] Examples: .0, 1.0, 123456789012345678 90.123456789012345678 A literal has *data type = DECIMAL* if it contains a period, and does not contain «E». DECIMAL literals may contain up to 38 digits; if there are more, then post-decimal digits may be subject to rounding. In earlier Tarantool versions literals containing periods were considered to be *NUMBER* or *DECIMAL* literals.

STRING literals: [quote] [character ...] [quote] Examples: 'ABC', 'AB''C' A literal has *data type type = STRING* if it is a sequence of zero or more characters enclosed in single quotes. The sequence '' (two single quotes in a row) is treated as ' (a single quote) when enclosed in quotes, that is, 'A''B' is interpreted as A'B.

VARBINARY literals: X|x [quote] [hexadecimal-digit-pair ...] [quote] Example: X'414243', which will be displayed as 'ABC'. A literal has *data type = VARBINARY* («variable-length binary») if it is the letter X followed by quotes containing pairs of hexadecimal digits, representing byte values.

MAP literals: [left curly bracket] key [colon] value [right curly bracket] Examples: {'a':1}, {1:'a'} A map literal is a pair of curly brackets (also called «braces») enclosing a STRING or INTEGER or UUID literal (called the map «key») followed by a colon followed by any type of literal (called the map «value»). This is a minimal form of a *MAP expression*.

ARRAY literals: [left square bracket] [literal] [right square bracket] Examples: [1], ['a'] An ARRAY literal is a literal value which is enclosed inside square brackets. This is a minimal form of an *ARRAY expression*.

Here are four ways to put non-ASCII characters,such as the Greek letter  alpha, in string literals: First make sure that your shell program is set to accept characters as UTF-8. A simple way to check is SELECT hex(cast('' as VARBINARY)); If the result is CEB1 – which is the hexadecimal value for the UTF-8 representation of  – it is good.

- (1) Simply enclose the character inside `'...'`, `''`
- (2) Find out what is the hexadecimal code for the UTF-8 representation of , and enclose that inside `X'...'`, then cast to STRING because `X'...'` literals are data type VARBINARY not STRING, `CAST(X'CEB1' AS STRING)`
- (3) Find out what is the Unicode code point for , and pass that to the *CHAR function*. `CHAR(945) /* remember that this is  as data type STRING not VARBINARY */`
- (4) Enclose statements inside double quotes and include Lua escapes, for example `box.execute("SELECT '\206\177';")`

One can use the concatenation operator `||` to combine characters made with any of these methods.

Limitations: (Issue#2344) * `LENGTH('A''B') = 3` which is correct, but on the Tarantool console the display from `SELECT A''B;` is `A''B`, which is misleading. * It is unfortunate that `X'41'` is a byte sequence which looks the same as `'A'`, but it is not the same. `box.execute("select 'A' < X'41';")` is not legal at the moment. This happens because `TYPEOF(X'41')` yields `'varbinary'`. Also it is illegal to say `UPDATE ... SET string_column = X'41'`, one must say `UPDATE ... SET string_column = CAST(X'41' AS STRING);`.

### Identifiers

All database objects – tables, triggers, indexes, columns, constraints, functions, collations – have identifiers. An identifier should begin with a letter or underscore (`'_'`) and should contain only letters, digits, dollar signs (`'$'`), or underscores. The maximum number of bytes in an identifier is between 64982 and 65000. For compatibility reasons, Tarantool recommends that an identifier should not have more than 30 characters.

Letters in identifiers do not have to come from the Latin alphabet, for example the Japanese syllabic  and the Cyrillic letter д are legal. But be aware that a Latin letter needs only one byte but a Cyrillic letter needs two bytes, so Cyrillic identifiers consume a tiny amount more space.

### Reserved words

Certain words are reserved and should not be used for identifiers. The simple rule is: if a word means something in Tarantool SQL syntax, do not try to use it for an identifier. The current list of reserved words is:

ALL ALTER ANALYZE AND ANY ARRAY AS ASC ASENSITIVE AUTOINCREMENT BEGIN BETWEEN BINARY BLOB BOOL BOOLEAN BOTH BY CALL CASE CAST CHAR CHARACTER CHECK COLLATE COLUMN COMMIT CONDITION CONNECT CONSTRAINT CREATE CROSS CURRENT CURRENT_DATE CURRENT_TIME CURRENT_TIMESTAMP CURRENT_USER CURSOR DATE DATETIME DEC DECIMAL DECLARE DEFAULT DEFERRABLE DELETE DENSE_RANK DESC DESCRIBE DETERMINISTIC DISTINCT DOUBLE DROP EACH ELSE ELSEIF END ESCAPE EXCEPT EXISTS EXPLAIN FALSE FETCH FLOAT FOR FOREIGN FROM FULL FUNCTION GET GRANT GROUP HAVING IF IMMEDIATE IN INDEX INNER INOUT INSENSITIVE INSERT INT INTEGER INTERSECT INTO IS ITERATE JOIN LEADING LEAVE LEFT LIKE LIMIT LOCALTIME LOCALTIMESTAMP LOOP MAP MATCH NATURAL NOT NULL NUM NUMBER NUMERIC OF ON OR ORDER OUT OUTER OVER PARTIAL PARTITION PRAGMA PRECISION PRIMARY PROCEDURE RANGE RANK READS REAL RECURSIVE REFERENCES REGEXP RELEASE RENAME REPEAT REPLACE RESIGNAL RETURN REVOKE RIGHT ROLLBACK ROW ROWS ROW_NUMBER SAVEPOINT SCALAR SELECT SENSITIVE SEQSCAN SESSION SET SIGNAL SIMPLE SMALLINT SPECIFIC SQL START STRING SYSTEM TABLE TEXT THEN TO TRAILING TRANSACTION TRIGGER TRIM TRUE TRUNCATE UNION UNIQUE UNKNOWN UNSIGNED UPDATE USER USING UUID VALUES VARBINARY VARCHAR VIEW WHEN WHENEVER WHERE WHILE WITH

Identifiers may be enclosed in double quotes. These are called quoted identifiers or «delimited identifiers» (unquoted identifiers may be called «regular identifiers»). The double quotes are not part of the identifier. A delimited identifier may be a reserved word and may contain any printable character. Tarantool converts letters in regular identifiers to upper case before it accesses the database, so for statements like `CREATE TABLE a (a INTEGER PRIMARY KEY);` or `SELECT a FROM a;` the table name is A and the column name is A. However, Tarantool does not convert delimited identifiers to upper case, so for statements like `CREATE TABLE "a" ("a" INTEGER PRIMARY KEY);` or `SELECT "a" FROM "a";` the table name is a and the column name is a. The sequence `""` is treated as `"` when enclosed in double quotes, that is, `"A""B"` is interpreted as `"A"B"`.

Examples: things, t45, journal_entries_for_2017, ддд, `"into"`

Внутри некоторых инструкций к идентификаторам можно добавлять квалификаторы для исключения двусмысленности. Квалификатор — это идентификатор объекта более высокого уровня, за которым следует точка. Например, к столбцу `column1` в таблице `table1` можно обратиться как `table1.column1`. Идентификатор, в том числе с добавленным квалификатором, — своего рода имя объекта. Например, в `SELECT table1.column1, table2.column1 FROM table1, table2;` соответствующие квалификаторы дают понять, что первый используемый столбец — это `column1` из `table1`, а второй — `column1` из `table2`.

The rules are sometimes relaxed for compatibility reasons, for example some non-letter characters such as $ and « are legal in regular identifiers. However, it is better to assume that rules are never relaxed.

The following are examples of legal and illegal identifiers.

```
_A1   -- legal, begins with underscore and contains underscore | letter | digit
1_A   -- illegal, begins with digit
A$« -- legal, but not recommended, try to stick with digits and letters and underscores
+ -- illegal, operator token
grant -- illegal, GRANT is a reserved word
"grant" -- legal, delimited identifiers may be reserved words
"_space" -- legal, but Tarantool already uses this name for a system space
"A"."X" -- legal, for columns only, inside statements where qualifiers may be necessary
'a' -- illegal, single quotes are for literals not identifiers
A12345678901234567890123456789O -- legal, identifiers can be long
ддд -- legal, and will be converted to upper case in identifiers
```

The following example shows that conversion to upper case affects regular identifiers but not delimited identifiers.

```
CREATE TABLE "q" ("q" INTEGER PRIMARY KEY);
SELECT * FROM q;
-- Result = "error: 'no such table: Q'.
```

### Operands

An operand is something that can be operated on. Literals and column identifiers are operands. So are NULL and DEFAULT.

NULL and DEFAULT are keywords which represent values whose data types are not known until they are assigned or compared, so they are known by the technical term «contextually typed value specifications». (Exception: for the non-standard statement «SELECT NULL FROM table-name;» NULL has data type BOOLEAN.)

### Operand data types

Every operand has a data type.

For literals, *as seen earlier*, the data type is usually determined by the format.

For identifiers, the data type is usually determined by the definition.

The usual determination may change because of context or because of *explicit casting*.

For some SQL data type names there are *aliases*. An alias may be used for data definition. For example VARCHAR(5) and TEXT are aliases of STRING and may appear in `CREATE TABLE table_name (column_name VARCHAR(5) PRIMARY KEY);` but Tarantool, if asked, will report that the data type of `column_name` is STRING.

For every SQL data type there is a corresponding NoSQL type, for example an SQL STRING is stored in a NoSQL space as *type = „string“*.

To avoid confusion in this manual, all references to SQL data type names are in upper case and all similar words which refer to NoSQL types or to other kinds of object are in lower case, for example:

- STRING is a data type name, but string is a general term;
- NUMBER — имя типа данных, а «numeric» и «числовое значение» — общие термины.

Although it is common to say that a VARBINARY value is a «binary string», this manual will not use that term and will instead say «byte sequence».

Here are all the SQL data types, their corresponding NoSQL types, their aliases, and minimum / maximum literal examples.

| SQL type | NoSQL type | Aliases | Minimum | Maximum |
|---|---|---|---|---|
| BOOLEAN | boolean (логический) | BOOL | FALSE | TRUE |
| INTEGER | integer (целое число) | INT | -9223372036854775808 | 18446744073709551615 |
| UNSIGNED | unsigned | (none) | 0 | 18446744073709551615 |
| DOUBLE | double (числа с двойной точностью) | (none) | -1.79769e308 | 1.79769e308 |
| NUMBER | number | (none) | -1.79769e308 | 1.79769e308 |
| DECIMAL | decimal | DEC | -99999999999999999999999999999999999999 | 99999999999999999999999999999999999999 |
| STRING | строка | TEXT, VARCHAR(n) | `''` | `'many-characters'` |
| VARBINARY | varbinary | (none) | `X''` | `X'many-hex-digits'` |
| UUID | uuid | (none) | 00000000-0000-0000-0000-000000000000 | ffffffff-ffff-ffff-dfff-ffffffffffff |
| DATETIME | *datetime* | (none) | | |
| INTERVAL | *interval* | (none) | | |
| SCALAR | (различные) | (none) | FALSE | максимальное значение UUID |
| MAP | map | (none) | `{}` | `{big-key:big-value}` |
| ARRAY | array | (none) | [] | `[many values]` |
| ANY | any | (none) | FALSE | `[many values]` |

BOOLEAN values are FALSE, TRUE, and UNKNOWN (which is the same as NULL). FALSE is less than TRUE.

Значения INTEGER — это числовые значения, которые не содержат десятичной точки и не представлены в экспоненциальной форме записи. Возможные значения: от `-2^63` до `+2^64`, а также `NULL`.

Значения UNSIGNED — это числовые значения, которые не содержат десятичной точки и не представлены в экспоненциальной форме записи. Возможные значения: от `0` до `+2^64`, а также `NULL`.

Значения DOUBLE — это числовые значения, которые содержат десятичную точку (например, `0.5`) или представлены в экспоненциальной форме записи (например, `5E-1`). Диапазон возможных значений соответствует стандарту IEEE 754 чисел с плавающей точкой, а также включает в себя `NULL`. Числа, выходящие за пределы диапазона DOUBLE, могут быть представлены как `-inf` или `inf`.

Значения NUMBER имеют тот же диапазон, что и значения DOUBLE, но могут быть и целыми числами. Для NUMBER не существует отдельного формата записи (значения типа `1.5` или `1E555` считаются DOUBLE), поэтому при необходимости используйте *CAST*, чтобы привести значение к типу NUMBER. См. также описание типа *„number“* в NoSQL. Арифметические операции и встроенные арифметические функции с типами NUMBER не поддерживаются начиная с версии Tarantool 2.10.1.

DECIMAL values can contain up to 38 digits on either side of a decimal point. and any arithmetic with DECIMAL values has exact results (arithmetic with DOUBLE values could have approximate results instead of exact results). Before Tarantool v. 2.10.0 there was no literal format for DECIMAL, so it was necessary to use *CAST* to insist that a numeric has data type DECIMAL, for example `CAST(1.1 AS DECIMAL)` or `CAST('99999999999999999999999999999999999999' AS DECIMAL)`. See the description of NoSQL type *„decimal“*. DECIMAL support in SQL was added in Tarantool version 2.10.1.

STRING values are any sequence of zero or more characters encoded with UTF-8, or NULL. The possible character values are the same as for the Unicode standard. Byte sequences which are not valid UTF-8 characters are allowed but not recommended. STRING literal values are enclosed within single quotes, for example `'literal'`. If the VARCHAR alias is used for column definition, it must include a maximum length, for example column_1 VARCHAR(40). However, the maximum length is ignored. The data-type may be followed by *[COLLATE collation-name]*.

VARBINARY values are any sequence of zero or more octets (bytes), or NULL. VARBINARY literal values are expressed as X followed by pairs of hexadecimal digits enclosed within single quotes, for example `X'0044'`. VARBINARY's NoSQL equivalent is `'varbinary'` but not character string – the MessagePack storage is MP_BIN (MsgPack binary).

Значения UUID (универсальные уникальные идентификаторы) — это 32 шестнадцатеричные цифры или NULL. Обычный формат UUID это строка, разделённая дефисами на пять групп в формате 8-4-4-4-12, например, `'000024ac-7ca6-4ab2-bd75-34742ac91213'`. В MessagePack (расширение MP_EXT) для хранения UUID требуется 16 байт. Значения UUID могут быть созданы с помощью *модуля uuid* Tarantool/NoSQL или с помощью функций UUID(). Поддержка UUID в SQL была добавлена в версии Tarantool 2.9.1.

DATETIME. Introduced in v. 2.10.0. A datetime table field can be created by using this type, which is semantically equivalent to the standard TIMESTAMP WITH TIME ZONE type.

```
tarantool> create table T2(d datetime primary key);
---
- row_count: 1
...

tarantool> insert into t2 values ('2022-01-01');
---
- null
- 'Type mismatch: can not convert string(''2022-01-01'') to datetime'
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> insert into t2 values (cast('2022-01-01' as datetime));
---
- row_count: 1
...

tarantool> select * from t2;
---
- metadata:
  - name: D
    type: datetime
  rows:
  - ['2022-01-01T00:00:00Z']
...
```

There is no implicit cast available from a string expression to a datetime expression (unlike convention used by majority of SQL vendors). In such cases, you need to use explicit cast from a string value to a datetime value (see the example above).

You can subtract datetime and datetime, datetime and interval, or add datetime and interval in any order (see examples of such arithmetics in the description of the *INTERVAL type*).

The built-in functions related to the DATETIME type are *DATE_PART()* and *NOW()*

INTERVAL. Introduced in v. 2.10.0. Similarly to the *DATETIME* type, you can define a column of the INTERVAL type.

```
tarantool> create table T(d datetime primary key, i interval);
---
- row_count: 1
...

tarantool> insert into T values (cast('2022-02-02T01:01' as datetime), cast({'year': 1,
↪'month': 1} as interval));
---
- row_count: 1
...

tarantool> select * from t;
---
- metadata:
  - name: D
    type: datetime
  - name: I
    type: interval
  rows:
  - ['2022-02-02T01:01:00Z', '+1 years, 1 months']
...
```

Unlike DATETIME, INTERVAL cannot be a part of an index.

There is no implicit cast available for conversions to an interval from a string or any other type. But there is explicit cast allowed from maps (see examples below).

Intervals can be used in arithmetic operations like + or - only with the datetime expression or another interval:

```
tarantool> select * from t
---
- metadata:
  - name: D
    type: datetime
  - name: I
    type: interval
  rows:
  - ['2022-02-02T01:01:00Z', '+1 years, 1 months']
...

tarantool> select d, d + i, d + cast({'year': 1, 'month': 2} as interval) from t
---
- metadata:
  - name: D
    type: datetime
  - name: COLUMN_1
    type: datetime
  - name: COLUMN_2
    type: datetime
  rows:
  - ['2022-02-02T01:01:00Z', '2023-03-02T01:01:00Z', '2023-04-02T01:01:00Z']
...

tarantool> select i + cast({'year': 1, 'month': 2} as interval) from t
---
- metadata:
  - name: COLUMN_1
    type: interval
  rows:
  - ['+2 years, 3 months']
...
```

There is the predefined list of known attributes for the map if you want to convert one to the INTERVAL expression:

- year
- month
- week
- day
- hour
- minute
- second
- nsec

```
tarantool> select cast({'year': 1, 'month': 1, 'week': 1, 'day': 1, 'hour': 1, 'min': 1,
↪'sec': 1} as interval)
---
- metadata:
  - name: COLUMN_1
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    type: interval
  rows:
  - ['+1 years, 1 months, 1 weeks, 1 days, 1 hours, 1 minutes, 1 seconds']
...

tarantool> \set language lua


tarantool> v = {year = 1, month = 1, week = 1, day = 1, hour = 1,
         >      min = 1, sec = 1, nsec = 1, adjust = 'none'}
---
...

tarantool> box.execute('select cast(#v as interval);', {{['#v'] = v}})

---
- metadata:
  - name: COLUMN_1
    type: interval
  rows:
  - ['+1 years, 1 months, 1 weeks, 1 days, 1 hours, 1 minutes, 1.000000001 seconds']
...
```

SCALAR может использоваться в *определениях столбцов*. Отдельные значения столбца также могут иметь тип SCALAR. Подробную информацию вы найдете в разделе *Определение столбцов — правила для типа данных SCALAR*. Вместе с этим типом данных может использоваться *[COLLATE название-сортировки]*. До версии Tarantool 2.10.1 отдельные значения столбцов могли иметь один из перечисленных выше типов: BOOLEAN, INTEGER, DOUBLE, DECIMAL, STRING, VARBINARY или UUID. Начиная с версии Tarantool 2.10.1 все значения в столбце SCALAR имеют тип SCALAR.

MAP values are key:value combinations which can be produced with *MAP expressions*. Maps cannot be used in arithmetic or comparison (except `IS [NOT] NULL`), and the only functions where they are allowed are *CAST*, *QUOTE*, *TYPEOF*, and functions involving NULL comparisons.

ARRAY values are lists which can be produced with *ARRAY expressions*. Arrays cannot be used in arithmetic or comparison (except `IS [NOT] NULL`), and the only functions where they are allowed are *CAST*, *QUOTE*, *TYPEOF*, and functions involving NULL comparisons.

ANY can be used for *column definitions* and the individual column values have type ANY. The difference between SCALAR and ANY is:

- SCALAR columns may not contain MAP or ARRAY values, but ANY columns may contain them.
- SCALAR values are comparable, while ANY values are not comparable.

Any value of any data type may be NULL. Ordinarily NULL will be cast to the data type of any operand it is being compared to or to the data type of the column it is in. If the data type of NULL cannot be determined from context, it is BOOLEAN.

Most of the SQL data types correspond to *Tarantool/NoSQL types* with the same name. In Tarantool versions before v. 2.10.0, There were also some Tarantool/NoSQL data types which had no corresponding SQL data types. In those versions, if Tarantool/SQL reads a Tarantool/NoSQL value of a type that has no SQL equivalent, Tarantool/SQL could treat it as NULL or INTEGER or VARBINARY. For example, `SELECT "flags" FROM "_vspace";` would return a column whose type is `'map'`. Such columns can only be manipulated in SQL by *invoking Lua functions*.

### Operators

An operator signifies what operation can be performed on operands.

Almost all operators are easy to recognize because they consist of one-character or two-character non-alphabetic tokens, except for six keyword operators (AND IN IS LIKE NOT OR).

Almost all operators are «dyadic», that is, they are performed on a pair of operands – the only operators that are performed on a single operand are NOT and ~ and (sometimes) -.

The result of an operation is a new operand. If the operator is a comparison operator then the result has data type BOOLEAN (TRUE or FALSE or UNKNOWN). Otherwise the result has the same data type as the original operands, except that: promotion to a broader type may occur to avoid overflow. Arithmetic with NULL operands will result in a NULL operand.

В списке операторов ниже пометка «(арифметическая операция)» указывает на то, что все операнды должны быть числовыми значениями (кроме NUMBER) и в результате тоже должно получиться число. Пометка «(сравнение)» указывает, что операнды должны иметь схожие типы данных и результат будет типа BOOLEAN. Пометка «(логическая операция)» указывает, что операнды должны быть типа BOOLEAN и результат также будет типа BOOLEAN. Если операция невозможна, выбрасывается исключение. Кроме того, существуют особые ситуации: их описание приводится после списка операторов. На месте указанных в примерах конкретных значений (литералов) могут с тем же успехом стоять идентификаторы столбцов.

Начиная с версии Tarantool 2.10.1 арифметические операнды не могут быть типа NUMBER.

- `+` addition (arithmetic)

  Add two numerics according to standard arithmetic rules. Example: `1 + 5`, result = 6.

- `-` subtraction (arithmetic)

  Subtract second numeric from first numeric according to standard arithmetic rules.

  Example: `1 - 5`, result = -4.

- `*` multiplication (arithmetic)

  Multiply two numerics according to standard arithmetic rules.

  Example: `2 * 5`, result = 10.

- `/` division (arithmetic)

  Divide second numeric into first numeric according to standard arithmetic rules. Division by zero is not legal. Division of integers always results in rounding toward zero, use *CAST* to DOUBLE or to DECIMAL to get non-integer results.

  Example: `5 / 2`, result = 2.

- `%` modulus (arithmetic)

  Divide second numeric into first numeric according to standard arithmetic rules. The result is the remainder. Starting with Tarantool version 2.10.1, operands must be INTEGER or UNSIGNED.

  Examples: `17 % 5`, result = 2; `-123 % 4`, result = -3.

- `<<` shift left (arithmetic)

  Shift the first numeric to the left N times, where N = the second numeric. For positive numerics, each 1-bit shift to the left is equivalent to multiplying times 2.

  Example: `5 << 1`, result = 10.

---

**Примечание:** Starting with Tarantool version 2.10.1, operands must be non-negative INTEGER or UNSIGNED.

---

- `>>` shift right (arithmetic)

  Shift the first numeric to the right N times, where N = the second numeric. For positive numerics, each 1-bit shift to the right is equivalent to dividing by 2.

  Example: `5 >> 1`, result = 2.

---

**Примечание:** Starting with Tarantool version 2.10.1, operands must be non-negative INTEGER or UNSIGNED.

---

- `&` and (arithmetic)

  Combine the two numerics, with 1 bits in the result if and only if both original numerics have 1 bits.

  Example: `5 & 4`, result = 4.

---

**Примечание:** Starting with Tarantool version 2.10.1, operands must be non-negative INTEGER or UNSIGNED.

---

- `|` or (arithmetic)

  Combine the two numerics, with 1 bits in the result if either original numeric has a 1 bit.

  Example: `5 | 2`, result = 7.

---

**Примечание:** Starting with Tarantool version 2.10.1, operands must be non-negative INTEGER or UNSIGNED.

---

- `~` negate (arithmetic), sometimes called bit inversion

  Change 0 bits to 1 bits, change 1 bits to 0 bits.

  Example: `~5`, result = -6.

---

**Примечание:** Starting with Tarantool version 2.10.1, the operand must be non-negative INTEGER or UNSIGNED.

---

- `<` less than (comparison)

  Return TRUE if the first operand is less than the second by arithmetic or collation rules.

  Example for numerics: `5 < 2`, result = FALSE

  Example for strings: `'C' < ' '`, result = FALSE

- `<=` less than or equal (comparison)

  Return TRUE if the first operand is less than or equal to the second by arithmetic or collation rules.

  Example for numerics: `5 <= 5`, result = TRUE

  Example for strings: `'C' <= 'B'`, result = FALSE

- `>` greater than (comparison)

  Return TRUE if the first operand is greater than the second by arithmetic or collation rules.

  Example for numerics: `5 > -5`, result = TRUE

  Example for strings: `'C' > '!'`, result = TRUE

- `>=` greater than or equal (comparison)

  Return TRUE if the first operand is greater than or equal to the second by arithmetic or collation rules.

  Example for numerics: `0 >= 0`, result = TRUE Example for strings: `'Z' >= ''`, result = FALSE

- `=` equal (assignment or comparison)

  After the word SET, «=» means the first operand gets the value from the second operand. In other contexts, «=» returns TRUE if operands are equal.

  Example for assignment: `... SET column1 = 'a';`

  Example for numerics: `0 = 0`, result = TRUE

  Example for strings: `'1' = '2 '`, result = FALSE

- `==` equal (assignment), or equal (comparison)

  This is a non-standard equivalent of *«= equal (assignment or comparison)»*.

- `<>` not equal (comparison)

  Return TRUE if the first operand is not equal to the second by arithmetic or collation rules.

  Example for strings: `'A' <> 'A '` is TRUE.

- `!=` not equal (comparison)

  This is a non-standard equivalent of *«<> not equal (comparison)»*.

- `[ , ]` (indexed access operator)

  Array example: `['a', 'b', 'c'] [2]` (returns `'b'`)

  Map example: `{'a' : 123, 7: 'asd'}['a']` (returns `123`)

  See also: *ARRAY index expression* and *MAP index expression*.

- `IS NULL` and `IS NOT NULL` (comparison)

  For IS NULL: Return TRUE if the first operand is NULL, otherwise return FALSE. Example: column1 IS NULL, result = TRUE if column1 contains NULL.

  For IS NOT NULL: Return FALSE if the first operand is NULL, otherwise return TRUE. Example: `column1 IS NOT NULL`, result = FALSE if column1 contains NULL.

- `LIKE` (comparison)

  Perform a comparison of two string operands. If the second operand contains `'_'`, the `'_'` matches any single character in the first operand. If the second operand contains `'%'`, the `'%'` matches 0 or more characters in the first operand. If it is necessary to search for either `'_'` or `'%'` within a string without treating it specially, an optional clause can be added, ESCAPE single-character-operand, for example `'abc_' LIKE 'abcX_' ESCAPE 'X'` is TRUE because `X'` means «following character is not special». Matching is also affected by the string's collation.

- `BETWEEN` (comparison)

  *x* `BETWEEN` *y* `AND` *z* is shorthand for *x* `>=` *y* `AND` *x* `<=` *z*.

- `NOT` negation (logic)

  Return TRUE if operand is FALSE return FALSE if operand is TRUE, else return UNKNOWN.

  Example: `NOT (1 > 1)`, result = TRUE.

- `IN` is equal to one of a list of operands (comparison)

  Return TRUE if first operand equals any of the operands in a parenthesized list.

  Example: `1 IN (2,3,4,1,7)`, result = TRUE.

- `AND` and (logic)

  Return TRUE if both operands are TRUE. Return UNKNOWN if both operands are UNKNOWN. Return UNKNOWN if one operand is TRUE and the other operand is UNKNOWN. Return FALSE if one operand is FALSE and the other operand is (UNKNOWN or TRUE or FALSE).

- `OR` or (logic)

  Return TRUE if either operand is TRUE. Return FALSE if both operands are FALSE. Return UNKNOWN if one operand is UNKNOWN and the other operand is (UNKNOWN or FALSE).

- `||` concatenate (string manipulation)

  Return the value of the first operand concatenated with the value of the second operand.

  Example: `'A' || 'B'`, result = `'AB'`.

The precedence of dyadic operators is:

```
||
* / %
+ -
<< >> & |
<  <= > >=
=  == != <> IS IS NOT IN LIKE
AND
OR
```

To ensure a desired precedence, use `()` parentheses.

### Special situations

If one of the operands has data type DOUBLE, Tarantool uses floating-point arithmetic. This means that exact results are not guaranteed and rounding may occur without warning. For example, 4.7777777777777778 = 4.7777777777777777 is TRUE.

The floating-point values inf and -inf are possible. For example, `SELECT 1e318, -1e318;` will return «inf, -inf». Arithmetic on infinite values may cause NULL results, for example `SELECT 1e318 - 1e318;` is NULL and `SELECT 1e318 * 0;` is NULL.

SQL operations never return the floating-point value -nan, although it may exist in data created by Tarantool's NoSQL. In SQL, -nan is treated as NULL.

В предыдущих версиях Tarantool строка преобразовывалась в числовое значение, если она использовалась с арифметическим оператором и преобразование было возможно. Например, результатом выражения `'7' + '7'` было 14. Для операций сравнения строка `'7'` преобразовывалась в значение 7. Это называется неявным приведением. Оно было применимо для значений типа STRING и всех числовых типов данных. Начиная с версии Tarantool 2.10 неявное приведение в числа больше не поддерживается.

Ограничения (подробнее в Issue#2346 на GitHub): * Некоторые ключевые слова, например MATCH и REGEXP, зарезервированы, но в текущих или будущих версиях Tarantool их пока не планируется использовать. * `99999999999999999 << 210` возвращает 0.

### Expressions

An expression is a chunk of syntax that causes return of a value. Expressions may contain literals, column-names, operators, and parentheses.

Therefore these are examples of expressions: `1`, `1 + 1 << 1`, `(1 = 2) OR 4 > 3`, `'x' || 'y' || 'z'`.

Also there are two expressions that involve keywords:

- `value IS [NOT] NULL`: determine whether value is (not) `NULL`.
- `CASE ... WHEN ... THEN ... ELSE ... END`: set a series of conditions.

### ARRAY expressions

Usage: `[ value ... ]`

Examples: `[1,2,3,4]`, `[1,[2,3],4]`, `['a', "column_1", uuid()]`

An expression has data type = ARRAY if it is a sequence of zero or more values enclosed in square brackets (`[` and `]`). Often the values in the sequence are called «elements». The element data type may be anything, including ARRAY – that is, ARRAYs may be nested. Different elements may have different types. The Lua equivalent type is „array“.

### MAP expressions

Usage: `{ key : value }`

Literal examples: `{'a':1}`, `{ "column_1" : X'1234' }`

Non-literal examples: `{"a":"a"}`, `{UUID(): (SELECT 1) + 1}`, `{1:'a123', 'two':uuid()}`

An expression has data type = MAP if it is enclosed in curly brackets (also called braces) `{` and `}` and contains a key for identification, then a colon `:`, then a value for what the key identifies. The key data type must be INTEGER or STRING or UUID. The value data type may be anything, including MAP – that is, MAPs may be nested. The Lua equivalent type is „map“ but the syntax is slightly different, for example the SQL value `{'a': 1}` is represented in Lua as `{a = 1}`.

### ARRAY index expression

Usage: `array-value [square bracket] index [square bracket]`

Example: `['a', 'b', 'c'] [2]` (this returns „b“)

As in other languages, an element of an array can be referenced with an integer inside square brackets. The returned value is of type ANY.

The `SELECT` query below retrieves all score values stored in the second position of the `scores` array field:

```
CREATE TABLE plays (user_id INTEGER PRIMARY KEY, scores ARRAY);
INSERT INTO plays VALUES (1, [23, 17, 55, 48]);
INSERT INTO plays VALUES (2, [12, 8, 20, 33]);
SELECT scores[2] FROM plays;
/* ---
  rows:
  - [17]
  - [8]
... */
```

### MAP index expression

Usage: `map-value [square bracket] index [square bracket]`

Example: `{'a' : 123, 7: 'asd'}['a']` (this returns 123). The returned value is of type ANY.

The `SELECT` query below retrieves all values stored in the `name` attribute of the `info` map field:

```
CREATE TABLE bands (id INTEGER PRIMARY KEY, info MAP);
INSERT INTO bands VALUES (1, {'name': 'The Beatles', 'year': 1960});
INSERT INTO bands VALUES (2, {'name': 'The Doors', 'year': 1965});
SELECT info['name'] FROM bands;
/* ---
  rows:
  - ['The Beatles']
  - ['The Doors']
... */
```

See also: *subquery*.

### Comparing and ordering

Определить, равны ли два значения или первое больше/меньше второго, помогают специальные правила. Они применяются при поиске, сортировке результатов в порядке возрастания значений в столбце, а также определении уникальности содержимого столбца. Результатом сравнения могут быть три значения типа BOOLEAN: `TRUE`, `FALSE` или `UNKNOWN`. В любом сравнении, где ни один из операндов не является `NULL`, операнды считаются различными, если результат сравнения равен `FALSE`. Любой набор операндов, где все операнды отличаются друг от друга, считается уникальным.

Сравнение двух числовых значений: * infinity = infinity вернет `TRUE`; * обычные числовые значения сравниваются по обычным арифметическим правилам.

When comparing any value to NULL: (for examples in this paragraph assume that column1 in table T contains {NULL, NULL, 1, 2}) * value comparison-operator NULL is UNKNOWN (not TRUE and not FALSE), which affects «WHERE condition» because the condition must be TRUE, and does not affect «CHECK (condition)» because the condition must be either TRUE or UNKNOWN. Therefore SELECT * FROM T WHERE column1 > 0 OR column1 < 0 OR column1 = 0; returns only {1,2}, and the table can have been created with CREATE TABLE T (... column1 INTEGER, CHECK (column1 >= 0)); * for any operations that contain the keyword DISTINCT, NULLs are not distinct. Therefore SELECT DISTINCT column1 FROM T; will return {NULL,1,2}. * for grouping, NULL values sort together. Therefore SELECT column1, COUNT(*) FROM T GROUP BY column1; will include a row {NULL, 2}. * for ordering, NULL values sort together and are less than non-NULL values. Therefore SELECT column1 FROM T ORDER

BY column1; returns {NULL, NULL, 1,2}. * for evaluating a UNIQUE constraint or UNIQUE index, any number of NULLs is okay. Therefore CREATE UNIQUE INDEX i ON T (column1); will succeed.

When comparing any value (except an ARRAY or MAP or ANY) to a SCALAR: * This is always legal, and the result depends on the underlying type of the value. For example, if COLUMN1 is defined as SCALAR, and a value in the column is „a“, then COLUMN1 < 5 is a legal comparison and the result is FALSE because numeric is less than STRING.

Сравнение числового значения со значением типа STRING: * Сравнение разрешено, если значение STRING можно явно привести к числовому.

When comparing a BOOLEAN to a BOOLEAN: TRUE is greater than FALSE.

When comparing a VARBINARY to a VARBINARY: * The numeric value of each pair of bytes is compared until the end of the byte sequences or until inequality. If two byte sequences are otherwise equal but one is longer, then the longer one is greater.

When comparing for the sake of eliminating duplicates: * This is usually signalled by the word DISTINCT, so it applies to SELECT DISTINCT, to set operators such as UNION (where DISTINCT is implied), and to aggregate functions such as AVG(DISTINCT). * Two operators are «not distinct» if they are equal to each other, or are both NULL * If two values are equal but not identical, for example 1.0 and 1.00, they are non-distinct and there is no way to specify which one will be eliminated * Values in primary-key or unique columns are distinct due to definition.

When comparing a STRING to a STRING: * Ordinarily collation is «binary», that is, comparison is done according to the numeric values of the bytes. This can be cancelled by adding a *COLLATE clause* at the end of either expression. So `'A' < 'a'` and `'a' < 'Ä'`, but `'A' COLLATE "unicode_ci" = 'a'` and `'a' COLLATE "unicode_ci" = 'Ä'`. * When comparing a column with a string literal, the column's defined collation is used. * Ordinarily trailing spaces matter. So `'a' = 'a '` is not TRUE. This can be cancelled by using the *TRIM(TRAILING ...)* function.

When comparing any value to an ARRAY or MAP or ANY: * The result is an error.

Ограничения: * Ожидается, что при работе с VARBINARY не будет применяться LIKE.

### Statements

Инструкция (statement) состоит из ключевых слов и выражений языка SQL, которые предписывают Tarantool выполнять какие-либо действия с базой данных. Инструкции начинаются с одного из ключевых слов: ALTER, ANALYZE, COMMIT, CREATE, DELETE, DROP, EXPLAIN, INSERT, PRAGMA, RELEASE, REPLACE, ROLLBACK, SAVEPOINT, SELECT, SET, START, TRUNCATE, UPDATE, VALUES или WITH. В конце инструкции ставится точка с запятой `;`, хотя это и не является обязательным.

A client sends a statement to the Tarantool server. The Tarantool server parses the statement and executes it. If there is an error, Tarantool returns an error message.

### List of legal statements

In alphabetical order, the following statements are legal.

*ALTER TABLE table-name [RENAME or ADD CONSTRAINT or DROP CONSTRAINT clauses];* ANALYZE [table-name]; – temporarily disabled in current version *COMMIT; CREATE [UNIQUE] INDEX [IF NOT EXISTS] index-name ON table-name (column-name [, column-name ...]); CREATE TABLE [IF NOT EXISTS] table-name (column-or-constraint-definition [, column-or-constraint-definition ...]) [WITH ENGINE = engine-name]; CREATE TRIGGER [IF NOT EXISTS] trigger-name BEFORE|AFTER INSERT|UPDATE|DELETE ON table-name FOR EACH ROW BEGIN dml-statement [, dml-statement ...] END; CREATE VIEW [IF NOT EXISTS] view-name [(column-name [, column-name ...])] AS select-statement | values-statement; DROP INDEX [IF EXISTS] index-name ON table-name; DROP TABLE [IF EXISTS] table-name; DROP TRIGGER [IF EXISTS] trigger-name; DROP VIEW [IF EXISTS] view-name; EXPLAIN explainable-statement; INSERT INTO table-name [(column-name [, column-name ...])] values-statement | select-statement; PRAGMA pragma-name[(value)]; RELEASE SAVEPOINT savepoint-name; REPLACE INTO table-name VALUES (expression [, expression ...]); ROLLBACK [TO [SAVEPOINT] savepoint-name]; SAVEPOINT savepoint-name; SELECT [DISTINCT|ALL] expression [, expression ...] FROM [SEQSCAN] table-name | joined-table-names [AS alias] [WHERE expression] [GROUP BY expression [, expression ...]] [HAVING expression] [ORDER BY expression] LIMIT expression [OFFSET expression]]; SET SESSION session-name = session-value; START TRANSACTION; TRUNCATE TABLE table-name; UPDATE table-name SET column-name=expression [,column-name=expression...] [WHERE expression]; VALUES (expression [, expression ...]; WITH [RECURSIVE] common-table-expression;*

### Data Type Conversion

Data type conversion, also called casting, is necessary for any operation involving two operands X and Y, when X and Y have different data types. Or, casting is necessary for assignment operations (when INSERT or UPDATE is putting a value of type X into a column defined as type Y). Casting can be «explicit» when a user uses the *CAST* function, or «implicit» when Tarantool does a conversion automatically.

The general rules are fairly simple: Assignments and operations involving NULL cause NULL or UNKNOWN results. For arithmetic, convert to the data type which can contain both operands and the result. For explicit casts, if a meaningful result is possible, the operation is allowed. For implicit casts, if a meaningful result is possible and the data types on both sides are either STRINGs or most numeric types (that is, are STRING or INTEGER or UNSIGNED or DOUBLE or DECIMAL but not NUMBER), the operation is sometimes allowed.

The specific situations in this chart follow the general rules:

| ~ | В BOOLEAN | В число | В STRING | В VARBINARY | В UUID |
|---|---|---|---|---|---|
| Из BOOLEAN | AAA | --- | A-- | --- | --- |
| Из числа | --- | SSA | A-- | --- | --- |
| Из STRING | S-- | S-- | AAA | A-- | S-- |
| Из VARBINARY | --- | --- | A-- | AAA | S-- |
| Из UUID | --- | --- | A-- | A-- | AAA |

Where each entry in the chart has 3 characters: Where A = Always allowed, S = Sometimes allowed, - = Never allowed. The first character of an entry is for explicit casts, the second character is for implicit casts for assignment, the third character is for implicit cast for comparison. So AAA = Always for explicit, Always for Implicit (assignment), Always for Implicit (comparison).

The S «Sometimes allowed» character applies for these special situations: From STRING To BOOLEAN

is allowed if UPPER(string-value) = `'TRUE'` or `'FALSE'`. From numeric to INTEGER or UNSIGNED is allowed for cast and assignment only if the result is not out of range, and the numeric has no post-decimal digits. From STRING to INTEGER or UNSIGNED or DECIMAL is allowed only if the string has a representation of a numeric, and the result is not out of range, and the numeric has no post-decimal digits. From STRING to DOUBLE or NUMBER is allowed only if the string has a representation of a numeric. From STRING to UUID is allowed only if the value is (8 hexadecimal digits) hyphen (4 hexadecimal digits) hyphen (4 hexadecimal digits) hyphen (4 hexadecimal digits) hyphen (12 hexadecimal digits), such as `'8e3b281b-78ad-4410-bfe9-54806a586a90'`. From VARBINARY to UUID is allowed only if the value is 16 bytes long, as in `X'8e3b281b78ad4410bfe954806a586a90'`.

The chart does not show To|From SCALAR because the conversions depend on the type of the value, not the type of the column definition. Explicit cast to SCALAR is always allowed.

The chart does not show To|From ARRAY or MAP or ANY because almost no conversions are possible. Explicit cast to ANY, or casting any value to its original data type, is legal, but that is all. This is a slight change: before Tarantool v. 2.10.0, it was legal to cast such values as VARBINARY. It is still possible to use arguments with these types in *QUOTE* functions, which is a way to convert them to STRINGs.

---

**Примечание:** Since version *2.4.1*, the NUMBER type is processed in the same way as the *number* type in NoSQL Tarantool.

Начиная с версии Tarantool 2.10.1 недопустимы некоторые преобразования, которые раньше были разрешены: Явное приведение числового типа к BOOLEAN. Явное приведение BOOLEAN к числовому типу. Неявное приведение NUMBER к другим числовым типам при выполнении арифметических или встроенных функций. Неявное приведение числового типа к STRING. Неявное приведение STRING к числовому типу.

---

Examples of casts, illustrating the situations in the chart:

Выполнение `CAST(TRUE AS STRING)` допустимо. В строке «Из BOOLEAN», столбце «В STRING» приведенной выше таблицы стоит значение `A--`, где буква `A` относится к явному приведению и означает «Always Allowed» — всегда разрешено. Таким образом, результатом операции будет `'TRUE'`.

Выполнение `UPDATE ... SET varbinary_column = 'A'` завершится ошибкой. В строке «Из STRING», столбце «В VARBINARY» приведенной выше таблицы стоит значение `A--`, где второй символ `-` относится к неявному приведению при присваивании и` означает «не разрешено». Результатом будет сообщение об ошибке.

Выражение `1.7E-1 > 0` допустимо. В строке «Из числа», столбце «В число» приведенной выше таблицы стоит значение `SSA`, где третья буква A относится к неявному приведению при сравнении и означает «Always Allowed» — всегда разрешено. Таким образом, результатом операции будет `'TRUE'`.

Выполнение `11 > '2'` завершится ошибкой. В строке «Из числа», столбце «В STRING» приведенной выше таблицы стоит значение `A--`, где третий символ `-` относится к неявному приведению при сравнении и` означает «не разрешено». Результатом операции будет сообщение об ошибке. Подробное объяснение приводится ниже.

Выполнение `CAST('5' AS INTEGER)` допустимо. В строке «Из STRING», столбце «В число» приведенной выше таблицы стоит значение `S--`, где первая буква `S` относится к явному приведению и означает «Sometimes allowed» — иногда разрешено. При этом приведение `CAST('5.5' AS INTEGER)` завершится ошибкой, поскольку это не целое число. Если число содержит цифры после десятичной точки, а целевой тип приведения — INTEGER или UNSIGNED, присвоение не будет выполнено.

### Implicit string/numeric cast

Примеры в этом разделе справедливы только для версий Tarantool до 2.10. Начиная с Tarantool 2.10 неявное приведение строк и числовых значений больше не допускается.

Приведение STRING к INTEGER/DOUBLE/NUMBER/UNSIGNED (любому числовому типу) и наоборот, выполняемое в ходе операции присвоения или сравнения, может требовать особых условий.

```
1 = '1' /* сравнение значения STRING с числовым значением */ UPDATE ... SET string_column
= 1 /* запись в STRING числового значения */
```

Для операций сравнения всегда выполняется приведение STRING к числовому значению. Поэтому `1e2 = '100'` вернет `TRUE`, как и `11 > '2'`. Если приведение не удалось, числовое значение считается меньше, чем значение типа STRING. Так что `1e400 < ''` тоже вернет `TRUE`. Исключение: для оператора BETWEEN приведение производится к типу данных первого и последнего операндов. Поэтому выражение `'66' BETWEEN 5 AND '7'` вернет `TRUE`.

Начиная с Tarantool *2.5.1* действует измененный алгоритм присваивания. В связи с этим неявные приведения строк к числам недопустимы. Например, `INSERT INTO t (integer_column) VALUES ('5');` выдаст ошибку.

Implicit cast does happen if STRINGS are used in arithmetic. Therefore `'5' / '5' = 1`. If the cast fails, then the result is an error. Therefore `'5' / ''` is an error.

Неявное приведение не производится, если числовые значения используются в конкатенации или в LIKE. Поэтому выражение `5 || '5'` недопустимо.

In the following examples, implicit cast does not happen for values in SCALAR columns: `DROP TABLE scalars; CREATE TABLE scalars (scalar_column SCALAR PRIMARY KEY); INSERT INTO scalars VALUES (11), ('2'); SELECT * FROM scalars WHERE scalar_column > 11; /* 0 rows. So 11 > '2'. */ SELECT * FROM scalars WHERE scalar_column < '2'; /* 1 row. So 11 < '2'. */ SELECT max(scalar_column) FROM scalars; /* 1 row: '2'. So 11 < '2'. */ SELECT sum(scalar_column) FROM scalars; /* 1 row: 13. So cast happened. */` These results are not affected by indexing, or by reversing the operands.

Implicit cast does NOT happen for *GREATEST()* or *LEAST()*. Therefore `LEAST('5',6)` is 6.

For function arguments: If the function description says that a parameter has a specific data type, and implicit assignment casts are allowed, then arguments which are not passed with that data type will be converted before the function is applied. For example, the *LENGTH()* function expects a STRING or VARBINARY, and INTEGER can be converted to STRING, therefore LENGTH(15) will return the length of `'15'`, that is, 2. But implicit cast sometimes does NOT happen for parameters. Therefore `ABS('5')` will cause an error message after Issue#4159 is fixed. However, *TRIM(5)* will still be legal.

Although it is not a requirement of the SQL standard, implicit cast is supposed to help compatibility with other DBMSs. However, other DBMSs have different rules about what can be converted (for example they may allow assignment of `'inf'` but disallow comparison with `'1e5'`). And, of course, it is not possible to be compatible with other DBMSs and at the same time support SCALAR, which other DBMSs do not have.

### 9.4.3 SQL statements and clauses

The Statements and Clauses guide shows all Tarantool/SQL statements" syntax and use.

| Heading | Введение |
|---|---|
| *Statements that change data definition* | *ALTER TABLE*, *CREATE TABLE*, *DROP TABLE*, *CREATE VIEW*, *DROP VIEW*, *CREATE INDEX*, *DROP INDEX*, *CREATE TRIGGER*, *DROP TRIGGER* |
| *Statements that change data* | *INSERT*, *UPDATE*, *DELETE*, *REPLACE*, *TRUNCATE*, *SET* |
| *Statements that retrieve data* | *SELECT*, *VALUES*, *PRAGMA*, *EXPLAIN* |
| *Statements for transactions* | *START TRANSACTION*, *COMMIT*, *SAVEPOINT*, *RELEASE SAVEPOINT*, *ROLLBACK* |
| *Functions* | For example *CAST(...)*, *LENGTH(...)*, *VERSION()* |

#### Statements that change data definition

##### ALTER TABLE

Syntax:

- `ALTER TABLE` *`table-name`* `RENAME TO` *`new-table-name`*`;`
- `ALTER TABLE` *`table-name`* `ADD COLUMN` *`column-name column-definition`*`;`
- `ALTER TABLE` *`table-name`* `ADD CONSTRAINT` *`constraint-name constraint-definition`*`;`
- `ALTER TABLE` *`table-name`* `DROP CONSTRAINT` *`constraint-name`*`;`
- `ALTER TABLE` *`table-name`* `ENABLE|DISABLE CHECK CONSTRAINT` *`constraint-name`*`;`

ALTER is used to change a table's name or a table's elements.

**Примеры:**

For renaming a table with `ALTER ... RENAME`, the *old-table* must exist, the *new-table* must not exist. Example: `-- renaming a table: ALTER TABLE t1 RENAME TO t2;`

For adding a *column* with `ADD COLUMN`, the table must exist, the table must be empty, the column name must be unique within the table. Example with a STRING column that must start with X:

```
ALTER TABLE t1 ADD COLUMN s4 STRING CHECK (s4 LIKE 'X%');
```

`ALTER TABLE ... ADD COLUMN` support was added in version *2.7.1*.

For adding a *table constraint* with `ADD CONSTRAINT`, the table must exist, the table must be empty, the constraint name must be unique within the table. Example with a *foreign-key constraint definition*: `ALTER TABLE t1 ADD CONSTRAINT fk_s1_t1_1 FOREIGN KEY (s1) REFERENCES t1;`

It is not possible to say `CREATE TABLE table_a ... REFERENCES table_b ...` if table `b` does not exist yet. This is a situation where `ALTER TABLE` is handy – users can `CREATE TABLE table_a` without the foreign key, then `CREATE TABLE table_b`, then `ALTER TABLE table_a ... REFERENCES table_b ...`.

```
-- adding a primary-key constraint definition:
-- This is unusual because primary keys are created automatically
-- and it is illegal to have two primary keys for the same table.
-- However, it is possible to drop a primary-key index, and this
-- is a way to restore the primary key if that happens.
ALTER TABLE t1 ADD CONSTRAINT "pk_unnamed_T1_1" PRIMARY KEY (s1);

-- adding a unique-constraint definition:
-- Alternatively, you can say CREATE UNIQUE INDEX unique_key ON t1 (s1);
ALTER TABLE t1 ADD CONSTRAINT "unique_unnamed_T1_2" UNIQUE (s1);

-- Adding a check-constraint definition:
ALTER TABLE t1 ADD CONSTRAINT "ck_unnamed_T1_1" CHECK (s1 > 0);
```

For `ALTER ... DROP CONSTRAINT`, it is only legal to drop a named constraint. (Tarantool generates the constraint names automatically if the user does not provide them.) Since version *2.4.1*, it is possible to drop any of the named table constraints, namely, PRIMARY KEY, UNIQUE, FOREIGN KEY, and CHECK.

To remove a unique constraint, use either `ALTER ... DROP CONSTRAINT` or *DROP INDEX*, which will drop the constraint as well.

```
-- dropping a constraint:
ALTER TABLE t1 DROP CONSTRAINT "fk_unnamed_JJ2_1";
```

For `ALTER ... ENABLE|DISABLE CHECK CONSTRAINT`, it is only legal to enable or disable a named constraint, and Tarantool only looks for names of check constraints. By default a constraint is enabled. If a constraint is disabled, then the check will not be performed.

```
-- disabling and re-enabling a constraint:
ALTER TABLE t1 DISABLE CHECK CONSTRAINT c;
ALTER TABLE t1 ENABLE CHECK CONSTRAINT c;
```

Limitations:

- It is not possible to drop a column.
- It is not possible to modify NOT NULL constraints or column properties DEFAULT and *data type*. However, it is possible to modify them with Tarantool/NOSQL, for example by calling *space_ object:format()* with a different `is_nullable` value.

### CREATE TABLE

Syntax:

`CREATE TABLE [IF NOT EXISTS]` *`table-name`* `(column-definition or table-constraint list) [WITH ENGINE =` *`string`*`];`

Create a new base table, usually called a «table».

**Примечание:** A table is a *base table* if it is created with CREATE TABLE and contains data in persistent storage.

A table is a *viewed table*, or just «view», if it is created with *CREATE VIEW* and gets its data from other views or from base tables.

The *table-name* must be an identifier which is valid according to the rules for identifiers, and must not be the name of an already existing base table or view.

The *column-definition* or *table-constraint* list is a comma-separated list of *column definitions* or *table constraint definitions*. Column definitions and table constraint definitions are sometimes called *table elements*.

Rules:

- A primary key is necessary; it can be specified with a table constraint PRIMARY KEY.
- There must be at least one column.
- When IF NOT EXISTS is specified, and there is already a table with the same name, the statement is ignored.
- When `WITH ENGINE = string` is specified, where `string` must be either „memtx“ or „vinyl“, the table is created with that *storage engine*. When this clause is not specified, the table is created with the default engine, which is ordinarily „memtx“ but may be changed by updating the *box.space._session_settings* system table..

Actions:

1. Tarantool evaluates each column definition and table-constraint, and returns an error if any of the rules is violated.
2. Tarantool makes a new definition in the schema.
3. Tarantool makes new indexes for PRIMARY KEY or UNIQUE constraints. A unique index name is created automatically.
4. Usually Tarantool effectively executes a *COMMIT* statement.

Примеры:

```
-- the simplest form, with one column and one constraint:
CREATE TABLE t1 (s1 INTEGER, PRIMARY KEY (s1));

-- you can see the effect of the statement by querying
-- Tarantool system spaces:
SELECT * FROM "_space" WHERE "name" = 'T1';
SELECT * FROM "_index" JOIN "_space" ON "_index"."id" = "_space"."id"
         WHERE "_space"."name" = 'T1';

-- variation of the simplest form, with delimited identifiers
-- and a bracketed comment:
CREATE TABLE "T1" ("S1" INT /* synonym of INTEGER */, PRIMARY KEY ("S1"));

-- two columns, one named constraint
CREATE TABLE t1 (s1 INTEGER, s2 STRING, CONSTRAINT pk_s1s2_t1_1 PRIMARY KEY (s1, s2));
```

Limitations:

- The maximum number of columns is 2000.

- The maximum length of a row depends on the *memtx_max_tuple_size* or *vinyl_max_tuple_size* configuration option.

### Column definition

Syntax:

`column-name data-type [, column-constraint]`

Define a column, which is a table element used in a *CREATE TABLE* statement.

The `column-name` must be an identifier which is valid according to the rules for identifiers.

Each `column-name` must be unique within a table.

### Column definition – data type

Every column has a data type: ANY or ARRAY or BOOLEAN or DECIMAL or DOUBLE or INTEGER or MAP or NUMBER or SCALAR or STRING or UNSIGNED or UUID or VARBINARY. The detailed description of data types is in the section *Operands*.

### Column definition – the rules for the SCALAR data type

The rules for the SCALAR data type were significantly changed in Tarantool version v. 2.10.0.

SCALAR is a «complex» data type, unlike all the other data types which are «primitive». Two column values in a SCALAR column can have two different primitive data types.

1. Any item defined as SCALAR has an underlying primitive type. For example, here:

   ```
   CREATE TABLE t (s1 SCALAR PRIMARY KEY);
   INSERT INTO t VALUES (55), ('41');
   ```

   the underlying primitive type of the item in the first row is INTEGER because literal 55 has data type INTEGER, and the underlying primitive type in the second row is STRING (the data type of a literal is always clear from its format).

   Примитивный тип элемента гораздо менее важен, чем его установленный тип. Tarantool может определить примитивный тип по тому, как MsgPack хранит его, но это уже детали реализации.

2. A SCALAR definition may not include a maximum length, as there is no suggested restriction.
3. A SCALAR definition may include a *COLLATE clause*, which affects any items whose primitive data type is STRING. The default collation is «binary».
4. Некоторые присваивания недопустимы, когда типы данных различаются, но разрешены, если речь об элементе типа SCALAR. Например, инструкция `UPDATE ... SET column1 = 'a'` выдаст ошибку, если столбец `column1` определен как INTEGER. Однако она может быть выполнена, если `column1` определен как SCALAR, при этом все значения типа INTEGER будут приведены к типу данных SCALAR.
5. There is no literal syntax which implies data type SCALAR.

6. Результат `TYPEOF(x)` — всегда `'scalar'` или `'NULL'`, а не базовый тип данных. Не существует функции, которая гарантированно возвращает базовый тип данных. Например, `TYPEOF(CAST(1 AS SCALAR));` возвращает `'scalar'`, а не `'integer'`.

7. Любая операция, которая требует неявного приведения элемента, определенного как SCALAR, завершится ошибкой во время выполнения. Например, если определение имеет вид:

```
CREATE TABLE t (s1 SCALAR PRIMARY KEY, s2 INTEGER);
```

   и единственная строка в таблице T содержит значение `s1 = 1`, то есть примитивный тип этого значения — INTEGER, то операция `UPDATE t SET s2 = s1;` все равно не разрешена.

8. For any dyadic operation that requires implicit casting for comparison, the syntax is legal and the operation will not fail at runtime. Take this situation: comparison with a primitive type VARBINARY and a primitive type STRING.

```
CREATE TABLE t (s1 SCALAR PRIMARY KEY);
INSERT INTO t VALUES (X'41');
SELECT * FROM t WHERE s1 > 'a';
```

   Такое сравнение корректно, поскольку Tarantool знает, в каком порядке стоят значения `X'41'` и `'a'` типа `'scalar'` в Tarantool/NoSQL. В этом случае примитивный тип имеет значение.

9. Результат операции *min/max* над столбцом, определенным как SCALAR, тоже будет иметь тип SCALAR. Пользователям необходимо заранее знать примитивный тип результата. Пример:

```
CREATE TABLE t (s1 INTEGER, s2 SCALAR PRIMARY KEY);
INSERT INTO t VALUES (1, X'44'), (2, 11), (3, 1E4), (4, 'a');
SELECT cast(min(s2) AS INTEGER), hex(cast(max(s2) as VARBINARY)) FROM t;
```

   The result is: `- - [11, '44',]`

   Это возможно только с правилами Tarantool/NoSQL для типа `scalar`. Однако вызов `SELECT SUM(s2)` не будет допустим: суммирование в этом случае потребует неявного приведения VARBINARY к числовому значению, что неразумно.

10. The result data type of a primitive combination is sometimes SCALAR although Tarantool in effect uses the primitive data type not the defined data type. (Here the word «combination» is used in the way that the standard document uses it for section «Result of data type combinations».) Therefore for `greatest(1E308, 'a', 0, X'00')` the result is X'00" but `typeof(greatest(1E308, 'a', 0, X'00')` is „scalar".

11. Объединение двух значений типа SCALAR иногда представляется примитивным типом. Например, `SELECT TYPEOF((SELECT CAST('a' AS SCALAR) UNION SELECT CAST('a' AS SCALAR)));` вернет `'string'`.

### Column definition – relation to NoSQL

All of the SQL data types except SCALAR correspond to *Tarantool/NoSQL types with the same name*. For example an SQL STRING is stored in a NoSQL space as type = „string".

Therefore specifying an SQL data type X determines that the storage will be in a space with a format column saying that the NoSQL type is „x".

The rules for that NoSQL type are applicable to the SQL data type.

If two items have SQL data types that have the same underlying type, then they are compatible for all assignment or comparison purposes.

If two items have SQL data types that have different underlying types, then the rules for explicit casts, or implicit (assignment) casts, or implicit (comparison) casts, apply.

Существует одно значение с плавающей точкой, которое не обрабатывается SQL: `-NaN` воспринимается как `NULL`, хотя его тип данных — `'double'`.

Before Tarantool v. 2.10.0, there were also some Tarantool/NoSQL data types which had no corresponding SQL data types. For example, `SELECT "flags" FROM "_vspace";` would return a column whose SQL data type is VARBINARY rather than MAP. Such columns can only be manipulated in SQL by *invoking Lua functions*.

### Column definition – column-constraint or default clause

The column-constraint or default clause may be as follows:

| Тип | Comment |
|---|---|
| NOT NULL | means «it is illegal to assign a NULL to this column» |
| PRIMARY KEY | explained in the *Table constraint definition* section |
| UNIQUE | explained in the *Table constraint definition* section |
| CHECK (expression) | explained in the *Table constraint definition* section |
| foreign-key-clause | explained in the *Table constraint definition for foreign keys* section |
| DEFAULT expression | means «if INSERT does not assign to this column then assign expression result to this column» – if there is no DEFAULT clause then DEFAULT NULL is assumed |

If column-constraint is PRIMARY KEY, this is a shorthand for a separate *table-constraint definition*: «PRIMARY KEY (column-name)».

If column-constraint is UNIQUE, this is a shorthand for a separate *table-constraint definition*: «UNIQUE (column-name)».

If column-constraint is CHECK, this is a shorthand for a separate *table-constraint definition*: «CHECK (expression)».

Columns defined with PRIMARY KEY are automatically NOT NULL.

To enforce some restrictions that Tarantool does not enforce automatically, add CHECK clauses, like these:

```
CREATE TABLE t ("smallint" INTEGER PRIMARY KEY CHECK ("smallint" <= 32767 AND "smallint"␣
↪>= -32768));
CREATE TABLE t ("shorttext" STRING PRIMARY KEY CHECK (length("shorttext") <= 10));
```

but this may cause inserts or updates to be slow.

### Column definition – examples

These are shown within *CREATE TABLE* statements. Data types may also appear in *CAST* functions.

```
-- the simple form with column-name and data-type
CREATE TABLE t (column1 INTEGER ...);
-- with column-name and data-type and column-constraint
CREATE TABLE t (column1 STRING PRIMARY KEY ...);
-- with column-name and data-type and collate-clause
CREATE TABLE t (column1 SCALAR COLLATE "unicode" ...);
```

```
-- with all possible data types and aliases
CREATE TABLE t
(column1 BOOLEAN, column2 BOOL,
 column3 INT PRIMARY KEY, column4 INTEGER,
 column5 DOUBLE,
 column6 NUMBER,
 column7 STRING, column8 STRING COLLATE "unicode",
 column9 TEXT, columna TEXT COLLATE "unicode_sv_s1",
 columnb VARCHAR(0), columnc VARCHAR(100000) COLLATE "binary",
 columnd UUID,
 columne VARBINARY,
 columnf SCALAR, columng SCALAR COLLATE "unicode_uk_s2",
 columnh DECIMAL,
 columni ARRAY,
 columnj MAP,
 columnk ANY);
```

```
-- with all possible column constraints and a default clause
CREATE TABLE t
(column1 INTEGER NOT NULL,
 column2 INTEGER PRIMARY KEY,
 column3 INTEGER UNIQUE,
 column4 INTEGER CHECK (column3 > column2),
 column5 INTEGER REFERENCES t,
 column6 INTEGER DEFAULT NULL);
```

### Table constraint definition

A table *constraint* restricts the data you can add to the table. If you try to insert invalid data on a column, Tarantool throws an error.

A table constraint has the following syntax:

```
[CONSTRAINT [name]] constraint_expression

constraint_expression:
  | PRIMARY KEY (column_name, ...)
  | UNIQUE (column_name, ...)
  | CHECK (expression)
  | FOREIGN KEY (column_name, ...) foreign_key_clause
```

Define a constraint, which is a table element used in a CREATE TABLE statement.

A constraint name must be an identifier that is valid according to the rules for identifiers. A constraint name must be unique within the table for a specific constraint type. For example, the CHECK and FOREIGN KEY constraints can have the same name.

#### PRIMARY KEY constraints

PRIMARY KEY constraints look like this:

```
PRIMARY KEY (column_name, ...)
```

There is a shorthand: specifying PRIMARY KEY in a *column definition*.

- Every table must have one and only one primary key.
- Primary-key columns are automatically `NOT NULL`.
- Primary-key columns are automatically indexed.
- Primary-key columns are unique. That means it is illegal to have two rows with the same values for the columns specified in the constraint.

#### Example 1: one-column primary key

1. Create an `author` table with the `id` primary key column:

   ```
   CREATE TABLE author (
       id INTEGER PRIMARY KEY,
       name STRING NOT NULL
   );
   ```

   Insert data into this table:

   ```
   INSERT INTO author VALUES (1, 'Leo Tolstoy'),
                             (2, 'Fyodor Dostoevsky');
   ```

2. On an attempt to add an author with the existing id, the following error is raised:

   ```
   INSERT INTO author VALUES (2, 'Alexander Pushkin');
   /*
   - Duplicate key exists in unique index "pk_unnamed_author_1" in space "author" with
     old tuple - [2, "Fyodor Dostoevsky"] and new tuple - [2, "Alexander Pushkin"]
   */
   ```

#### Example 2: two-column primary key

1. Create a `book` table with the primary key defined on two columns:

   ```
   CREATE TABLE book (
       id INTEGER,
       title STRING NOT NULL,
       PRIMARY KEY (id, title)
   );
   ```

   Insert data into this table:

   ```
   INSERT INTO book VALUES (1, 'War and Peace'),
                           (2, 'Crime and Punishment');
   ```

2. On an attempt to add the existing book, the following error is raised:

```
INSERT INTO book VALUES (2, 'Crime and Punishment');
/*
- Duplicate key exists in unique index "pk_unnamed_book_1" in space "BOOK" with old
  tuple - [2, "Crime and Punishment"] and new tuple - [2, "Crime and Punishment"]
*/
```

PRIMARY KEY with the AUTOINCREMENT modifier may be specified in one of two ways:

- In a column definition after the words PRIMARY KEY, as in `CREATE TABLE t (c INTEGER PRIMARY KEY AUTOINCREMENT);`
- In a PRIMARY KEY (column-list) after a column name, as in `CREATE TABLE t (c INTEGER, PRIMARY KEY (c AUTOINCREMENT));`

When AUTOINCREMENT is specified, the column must be a primary-key column and it must be INTEGER or UNSIGNED.

Only one column in the table may be autoincrement. However, it is legal to say `PRIMARY KEY (a, b, c AUTOINCREMENT)` – in that case, there are three columns in the primary key but only the third column (`c`) is AUTOINCREMENT.

As the name suggests, values in an autoincrement column are automatically incremented. That is: if a user inserts NULL in the column, then the stored value will be the smallest non-negative integer that has not already been used. This occurs because autoincrement columns are associated with *sequences*.

**UNIQUE constraints**

UNIQUE constraints look like this:

```
UNIQUE (column_name, ...)
```

There is a shorthand: specifying UNIQUE in a *column definition*.

Unique constraints are similar to primary-key constraints, except that:

- A table may have any number of unique keys, and unique keys are not automatically NOT NULL.
- Unique columns are automatically indexed.
- Unique columns are unique. That means it is illegal to have two rows with the same values in the unique-key columns.

**Example 1: one-column unique constraint**

1. Create an `author` table with the unique `name` column:

   ```
   CREATE TABLE author (
       id INTEGER PRIMARY KEY,
       name STRING UNIQUE
   );
   ```

   Insert data into this table:

   ```
   INSERT INTO author VALUES (1, 'Leo Tolstoy'),
                             (2, 'Fyodor Dostoevsky');
   ```

2. On an attempt to add an author with the same name, the following error is raised:

```
INSERT INTO author VALUES (3, 'Leo Tolstoy');
/*
- Duplicate key exists in unique index "unique_unnamed_author_2" in space "author"
  with old tuple - [1, "Leo Tolstoy"] and new tuple - [3, "Leo Tolstoy"]
*/
```

**Example 2: two-column unique constraint**

1. Create a `book` table with the unique constraint defined on two columns:

```
CREATE TABLE book (
    id INTEGER PRIMARY KEY,
    title STRING NOT NULL,
    author_id INTEGER UNIQUE,
    UNIQUE (title, author_id)
);
```

   Insert data into this table:

```
INSERT INTO book VALUES (1, 'War and Peace', 1),
                        (2, 'Crime and Punishment', 2);
```

2. On an attempt to add a book with duplicated values, the following error is raised:

```
INSERT INTO book VALUES (3, 'War and Peace', 1);
/*
- Duplicate key exists in unique index "unique_unnamed_book_2" in space "book" with
  old tuple - [1, "War and Peace", 1] and new tuple - [3, "War and Peace", 1]
*/
```

### CHECK constraints

The CHECK constraint is used to limit the value range that a column can store. CHECK constraints look like this:

```
CHECK (expression)
```

There is a shorthand: specifying CHECK in a *column definition*.

The expression may be anything that returns a BOOLEAN result = TRUE or FALSE or UNKNOWN. The expression may not contain a *subquery*. If the expression contains a column name, the column must exist in the table. If a CHECK constraint is specified, the table must not contain rows where the expression is FALSE. (The table may contain rows where the expression is either TRUE or UNKNOWN.) Constraint checking may be stopped with *ALTER TABLE ... DISABLE CHECK CONSTRAINT* and restarted with ALTER TABLE ... ENABLE CHECK CONSTRAINT.

**Example**

1. Create an `author` table with the `name` column that should contain values longer than 4 characters:

```
CREATE TABLE author (
    id INTEGER PRIMARY KEY,
    name STRING,
    CONSTRAINT check_name_length CHECK (CHAR_LENGTH(name) > 4)
);
```

   Insert data into this table:

```
INSERT INTO author VALUES (1, 'Leo Tolstoy'),
                          (2, 'Fyodor Dostoevsky');
```

2. On an attempt to add an author with a name shorter than 5 characters, the following error is raised:

```
INSERT INTO author VALUES (3, 'Alex');
/*
- Check constraint 'check_name_length' failed for a tuple
*/
```

### Table constraint definition for foreign keys

A *foreign key* is a *constraint* that can be used to enforce data integrity across related tables. A foreign key constraint is defined on the child table that references the parent table's column values.

Foreign key constraints look like this:

```
FOREIGN KEY (referencing_column_name, ...)
    REFERENCES referenced_table_name (referenced_column_name, ...)
```

You can also add a reference in a *column definition*:

```
referencing_column_name column_definition
    REFERENCES referenced_table_name(referenced_column_name)
```

---

**Примечание:** Since *2.11.0*, the following referencing options aren't supported anymore:

- The `ON UPDATE` and `ON DELETE` triggers. The `RESTRICT` trigger action is used implicitly.
- The `MATCH` subclause. `MATCH FULL` is used implicitly.
- `DEFERRABLE` constraints. The `INITIALLY IMMEDIATE` constraint check time rule is used implicitly.

---

Note that a referenced column should meet one of the following requirements:

- A referenced column is a PRIMARY KEY column.
- A referenced column has a UNIQUE constraint.
- A referenced column has a UNIQUE index.

Note that before the *2.11.0* version, an index existence for the referenced columns is checked when creating a constraint (for example, using `CREATE TABLE` or `ALTER TABLE`). Starting with 2.11.0, this check is weakened and the existence of an index is checked during data insertion.

**Example**

This example shows how to create a relation between the parent and child tables through a single-column foreign key:

1. First, create a parent `author` table:

```
CREATE TABLE author (
    id INTEGER PRIMARY KEY,
    name STRING NOT NULL
);
```

Insert data into this table:

```
INSERT INTO author VALUES (1, 'Leo Tolstoy'),
                          (2, 'Fyodor Dostoevsky');
```

2. Create a child `book` table whose `author_id` column references the `id` column from the `author` table:

```
CREATE TABLE book (
    id INTEGER PRIMARY KEY,
    title STRING NOT NULL,
    author_id INTEGER NOT NULL UNIQUE,
    FOREIGN KEY (author_id)
        REFERENCES author (id)
);
```

Alternatively, you can add a reference in a column definition:

```
CREATE TABLE book (
    id INTEGER PRIMARY KEY,
    title STRING NOT NULL,
    author_id INTEGER NOT NULL UNIQUE REFERENCES author(id)
);
```

Insert data to the `book` table:

```
INSERT INTO book VALUES (1, 'War and Peace', 1),
                        (2, 'Crime and Punishment', 2);
```

3. Check how the created foreign key constraint enforces data integrity. The following error is raised on an attempt to insert a new book with the `author_id` value that doesn't exist in the parent `author` table:

```
INSERT INTO book VALUES (3, 'Eugene Onegin', 3);
/*
- 'Foreign key constraint ''fk_unnamed_book_1'' failed: foreign tuple was not found'
*/
```

On an attempt to delete an author that already has books in the `book` table, the following error is raised:

```
DELETE FROM author WHERE id = 2;
/*
- 'Foreign key ''fk_unnamed_book_1'' integrity check failed: tuple is referenced'
*/
```

### DROP TABLE

Syntax:

```
DROP TABLE [IF EXISTS] table-name ;
```

Drop a table.

The *table-name* must identify a table that was created earlier with the *CREATE TABLE statement*.

Rules:

- If there is a view that references the table, the drop will fail. Please drop the referencing view with *DROP VIEW* first.
- If there is a foreign key that references the table, the drop will fail. Please drop the referencing constraint with *ALTER TABLE ... DROP* first.

Actions:

1. Tarantool returns an error if the table does not exist and there is no `IF EXISTS` clause.
2. The table and all its data are dropped.
3. All indexes for the table are dropped.
4. All triggers for the table are dropped.
5. Usually Tarantool effectively executes a *COMMIT* statement.

Примеры:

```
-- the simple case:
DROP TABLE t31;
-- with an IF EXISTS clause:
DROP TABLE IF EXISTS t31;
```

See also: *DROP VIEW*.

### CREATE VIEW

Syntax:

```
CREATE VIEW [IF NOT EXISTS] view-name [(column-list)] AS subquery;
```

Create a new viewed table, usually called a «view».

The *view-name* must be valid according to the rules for identifiers.

The optional *column-list* must be a comma-separated list of names of columns in the view.

The syntax of the subquery must be the same as the syntax of a *SELECT statement*, or of a VALUES clause.

Rules:

- There must not already be a base table or view with the same name as *view-name*.
- If *column-list* is specified, the number of columns in *column-list* must be the same as the number of columns in the *select list* of the subquery.

Actions:

1. Tarantool will throw an error if a rule is violated.
2. Tarantool will create a new persistent object with *column-names* equal to the names in the *column-list* or the names in the subquery's *select list*.
3. Usually Tarantool effectively executes a *COMMIT* statement.

Примеры:

```
-- the simple case:
CREATE VIEW v AS SELECT column1, column2 FROM t;
-- with a column-list:
CREATE VIEW v (a,b) AS SELECT column1, column2 FROM t;
```

Limitations:

- It is not possible to insert or update or delete from a view, although sometimes a possible substitution is to *create an INSTEAD OF trigger*.

### DROP VIEW

Syntax:

```
DROP VIEW [IF EXISTS] view-name ;
```

Drop a view.

The *view-name* must identify a view that was created earlier with the *CREATE VIEW statement*.

Rules: none

Actions:

1. Tarantool returns an error if the view does not exist and there is no `IF EXISTS` clause.
2. The view is dropped.
3. All triggers for the view are dropped.
4. Usually Tarantool effectively executes a *COMMIT* statement.

Примеры:

```
-- the simple case:
DROP VIEW v31;
-- with an IF EXISTS clause:
DROP VIEW IF EXISTS v31;
```

See also: *DROP TABLE*.

### CREATE INDEX

Syntax:

```
CREATE [UNIQUE] INDEX [IF NOT EXISTS] index-name ON table-name (column-list);
```

Create an index.

The *index-name* must be valid according to the rules for identifiers.

The *table-name* must refer to an existing table.

The *column-list* must be a comma-separated list of names of columns in the table.

Rules:

- There must not already be, for the same table, an index with the same name as *index-name*. But there may already be, for a different table, an index with the same name as *index-name*.
- The maximum number of indexes per table is 128.

Actions:

1. Tarantool will throw an error if a rule is violated.
2. If the new index is UNIQUE, Tarantool will throw an error if any row exists with columns that have duplicate values.
3. Tarantool will create a new index.
4. Usually Tarantool effectively executes a *COMMIT* statement.

Automatic indexes:

Indexes may be created automatically for columns mentioned in the PRIMARY KEY or UNIQUE clauses of a CREATE TABLE statement. If an index was created automatically, then the *index-name* has four parts:

1. `pk` if this is for a PRIMARY KEY clause, `unique` if this is for a UNIQUE clause;
2. `_unnamed_`;
3. the name of the table;
4. `_` and an ordinal number; the first index is 1, the second index is 2, and so on.

For example, after `CREATE TABLE t (s1 INTEGER PRIMARY KEY, s2 INTEGER, UNIQUE (s2));` there are two indexes named `pk_unnamed_T_1` and `unique_unnamed_T_2`. You can confirm this by saying `SELECT * FROM "_index";` which will list all indexes on all tables. There is no need to say `CREATE INDEX` for columns that already have automatic indexes.

Примеры:

```
-- the simple case
CREATE INDEX idx_column1_t_1 ON t (column1);
-- with IF NOT EXISTS clause
CREATE INDEX IF NOT EXISTS idx_column1_t_1 ON t (column1);
-- with UNIQUE specifier and more than one column
CREATE UNIQUE INDEX idx_unnamed_t_1 ON t (column1, column2);
```

Dropping an automatic index created for a unique constraint will drop the unique constraint as well.

### DROP INDEX

Syntax:

```
DROP INDEX [IF EXISTS] index-name ON table-name;
```

The *index-name* must be the name of an existing index, which was created with *CREATE INDEX*. Or, the *index-name* must be the name of an index that was created automatically due to a PRIMARY KEY or UNIQUE clause in the *CREATE TABLE* statement. To see what a table's indexes are, use *PRAGMA index_list(table-name);*.

Rules: none

Actions:

1. Tarantool throws an error if the index does not exist, or is an automatically created index.
2. Tarantool will drop the index.
3. Usually Tarantool effectively executes a *COMMIT* statement.

Пример:

```
-- the simplest form:
DROP INDEX idx_unnamed_t_1 ON t;
```

### CREATE TRIGGER

Syntax:

```
CREATE TRIGGER [IF NOT EXISTS] trigger-name BEFORE|AFTER|INSTEAD OF DELETE|INSERT|UPDATE
ON table-name FOR EACH ROW [WHEN search-condition] BEGIN delete-statement
| insert-statement | replace-statement | select-statement | update-statement;
[delete-statement | insert-statement | replace-statement | select-statement |
update-statement; ...] END;
```

The *trigger-name* must be valid according to the rules for identifiers.

If the trigger action time is BEFORE or AFTER, then the *table-name* must refer to an existing base table.

If the trigger action time is INSTEAD OF, then the *table-name* must refer to an existing view.

Rules:

- There must not already be a trigger with the same name as *trigger-name*.
- Triggers on different tables or views share the same namespace.
- The statements between BEGIN and END should not refer to the *table-name* mentioned in the ON clause.
- The statements between BEGIN and END should not contain an *INDEXED BY* clause.

SQL triggers are not activated by Tarantool/NoSQL requests. This will change in a future version.

On a *replica*, effects of trigger execution are applied, and the SQL triggers themselves are not activated upon replication events.

NoSQL triggers are activated both on replica and master, thus if you have a *NoSQL trigger* on a replica, it is activated when applying effects of an SQL trigger.

Actions:

1. Tarantool will throw an error if a rule is violated.
2. Tarantool will create a new trigger.
3. Usually Tarantool effectively executes a *COMMIT* statement.

Примеры:

```
-- the simple case:
CREATE TRIGGER stores_before_insert BEFORE INSERT ON stores FOR EACH ROW
  BEGIN DELETE FROM warehouses; END;
-- with IF NOT EXISTS clause:
CREATE TRIGGER IF NOT EXISTS stores_before_insert BEFORE INSERT ON stores FOR EACH ROW
  BEGIN DELETE FROM warehouses; END;
-- with FOR EACH ROW and WHEN clauses:
CREATE TRIGGER stores_before_insert BEFORE INSERT ON stores FOR EACH ROW WHEN a=5
  BEGIN DELETE FROM warehouses; END;
-- with multiple statements between BEGIN and END:
CREATE TRIGGER stores_before_insert BEFORE INSERT ON stores FOR EACH ROW
  BEGIN DELETE FROM warehouses; INSERT INTO inventories VALUES (1); END;
```

### Trigger extra clauses

- `UPDATE OF column-list`

  After BEFORE|AFTER UPDATE it is optional to add `OF column-list`. If any of the columns in *column-list* is affected at the time the row is processed, then the trigger will be activated for that row. For example:

  ```
  CREATE TRIGGER table1_before_update
   BEFORE UPDATE  OF column1, column2 ON table1
   FOR EACH ROW
   BEGIN UPDATE table2 SET column1 = column1 + 1; END;
  UPDATE table1 SET column3 = column3 + 1; -- Trigger will not be activated
  UPDATE table1 SET column2 = column2 + 0; -- Trigger will be activated
  ```

- `WHEN`

  After *table-name* FOR EACH ROW it is optional to add [`WHEN expression`]. If the expression is true at the time the row is processed, only then will the trigger will be activated for that row. For example:

  ```
  CREATE TRIGGER table1_before_update BEFORE UPDATE ON table1 FOR EACH ROW
   WHEN (SELECT COUNT(*) FROM table1) > 1
   BEGIN UPDATE table2 SET column1 = column1 + 1; END;
  ```

  This trigger will not be activated unless there is more than one row in `table1`.

- `OLD and NEW`

  The keywords OLD and NEW have special meaning in the context of trigger action:

  – OLD.column-name refers to the value of *column-name* before the change.

  – NEW.column-name refers to the value of *column-name* after the change.

  Пример:

```
CREATE TABLE table1 (column1 STRING, column2 INTEGER PRIMARY KEY);
CREATE TABLE table2 (column1 STRING, column2 STRING, column3 INTEGER PRIMARY KEY);
INSERT INTO table1 VALUES ('old value', 1);
INSERT INTO table2 VALUES ('', '', 1);
CREATE TRIGGER table1_before_update BEFORE UPDATE ON table1 FOR EACH ROW
 BEGIN UPDATE table2 SET column1 = old.column1, column2 = new.column1; END;
UPDATE table1 SET column1 = 'new value';
SELECT * FROM table2;
```

  At the beginning of the UPDATE for the single row of `table1`, the value in `column1` is „old value“ – so that is what is seen as `old.column1`.

  At the end of the UPDATE for the single row of `table1`, the value in `column1` is „new value“ – so that is what is seen as `new.column1`. (OLD and NEW are qualifiers for `table1`, not `table2`.)

  Therefore, `SELECT * FROM table2;` returns `['old value', 'new value']`.

  `OLD.column-name` does not exist for an INSERT trigger.

  `NEW.column-name` does not exist for a DELETE trigger.

  OLD and NEW are read-only; you cannot change their values.

- Deprecated or illegal statements:

  It is illegal for the trigger action to include a qualified column reference other than `OLD.column-name` or `NEW.column-name`. For example, `CREATE TRIGGER ... BEGIN UPDATE table1 SET table1.column1 = 5; END;` is illegal.

  It is illegal for the trigger action to include statements that include a *WITH clause*, a DEFAULT VALUES clause, or an *INDEXED BY* clause.

  It is usually not a good idea to have a trigger on `table1` which causes a change on `table2`, and at the same time have a trigger on `table2` which causes a change on `table1`. For example:

```
CREATE TRIGGER table1_before_update
 BEFORE UPDATE ON table1
 FOR EACH ROW
 BEGIN UPDATE table2 SET column1 = column1 + 1; END;
CREATE TRIGGER table2_before_update
 BEFORE UPDATE ON table2
 FOR EACH ROW
 BEGIN UPDATE table1 SET column1 = column1 + 1; END;
```

  Luckily `UPDATE table1 ...` will not cause an infinite loop, because Tarantool recognizes when it has already updated so it will stop. However, not every DBMS acts this way.

### Trigger activation

These are remarks concerning trigger activation.

Standard terminology:

- «trigger action time» = BEFORE or AFTER or INSTEAD OF
- «trigger event» = INSERT or DELETE or UPDATE
- «triggered statement» = BEGIN ... DELETE|INSERT|REPLACE|SELECT|UPDATE ... END
- «triggered when clause» = WHEN search-condition
- «activate» = execute a triggered statement
- some vendors use the word «fire» instead of «activate»

If there is more than one trigger for the same trigger event, Tarantool may execute the triggers in any order.

It is possible for a triggered statement to cause activation of another triggered statement. For example, this is legal:

```
CREATE TRIGGER t1_before_delete BEFORE DELETE ON t1 FOR EACH ROW BEGIN DELETE FROM t2;␣
↪END;
CREATE TRIGGER t2_before_delete BEFORE DELETE ON t2 FOR EACH ROW BEGIN DELETE FROM t3;␣
↪END;
```

Activation occurs FOR EACH ROW, not FOR EACH STATEMENT. Therefore, if no rows are candidates for insert or update or delete, then no triggers are activated.

The BEFORE trigger is activated even if the trigger event fails.

If an UPDATE trigger event does not make a change, the trigger is activated anyway. For example, if row 1 `column1` contains `'a'`, and the trigger event is `UPDATE ... SET column1 = 'a';`, the trigger is activated.

The triggered statement may refer to a function: `RAISE(FAIL, error-message)`. If a triggered statement invokes a `RAISE(FAIL, error-message)` function, or if a triggered statement causes an error, then statement execution stops immediately.

The triggered statement may refer to column values within the rows being changed. in this case:

- The row «as of before» the change is called the «old» row (which makes sense only for UPDATE and DELETE statements).
- The row «as of after» the change is called the «new» row (which makes sense only for UPDATE and INSERT statements).

This example shows how an INSERT can be done to a view by referring to the «new» row:

```
CREATE TABLE t (s1 INTEGER PRIMARY KEY, s2 INTEGER);
CREATE VIEW v AS SELECT s1, s2 FROM t;
CREATE TRIGGER v_instead_of INSTEAD OF INSERT ON v
  FOR EACH ROW
  BEGIN INSERT INTO t VALUES (new.s1, new.s2); END;
INSERT INTO v VALUES (1, 2);
```

Ordinarily saying `INSERT INTO view_name ...` is illegal in Tarantool, so this is a workaround.

It is possible to generalize this so that all data-change statements on views will change the base tables, provided that the view contains all the columns of the base table, and provided that the triggers refer to those columns when necessary, as in this example:

```
CREATE TABLE base_table (primary_key_column INTEGER PRIMARY KEY, value_column INTEGER);
CREATE VIEW viewed_table AS SELECT primary_key_column, value_column FROM base_table;
CREATE TRIGGER viewed_table_instead_of_insert INSTEAD OF INSERT ON viewed_table FOR EACH␣
↪ROW
  BEGIN
    INSERT INTO base_table VALUES (new.primary_key_column, new.value_column); END;
CREATE TRIGGER viewed_table_instead_of_update INSTEAD OF UPDATE ON viewed_table FOR EACH␣
↪ROW
  BEGIN
    UPDATE base_table
    SET primary_key_column = new.primary_key_column, value_column = new.value_column
    WHERE primary_key_column = old.primary_key_column; END;
CREATE TRIGGER viewed_table_instead_of_delete INSTEAD OF DELETE ON viewed_table FOR EACH␣
↪ROW
  BEGIN
    DELETE FROM base_table WHERE primary_key_column = old.primary_key_column; END;
```

When INSERT or UPDATE or DELETE occurs for table `X`, Tarantool usually operates in this order (a basic scheme):

```
For each row
  Perform constraint checks
  For each BEFORE trigger that refers to table X
    Check that the trigger's WHEN condition is true.
    Execute what is in the triggered statement.
  Insert or update or delete the row in table X.
  Perform more constraint checks
  For each AFTER trigger that refers to table X
    Check that the trigger's WHEN condition is true.
    Execute what is in the triggered statement.
```

However, Tarantool does not guarantee execution order when there are multiple constraints, or multiple triggers for the same event (including NoSQL *on_replace triggers* or SQL *INSTEAD OF triggers* that affect a view of table `X`).

The maximum number of trigger activations per statement is 32.

### INSTEAD OF triggers

A trigger which is created with the clause `INSTEAD OF` *`INSERT|UPDATE|DELETE`* `ON` *`view-name`* is an INSTEAD OF trigger. For each affected row, the trigger action is performed «instead of» the INSERT or UPDATE or DELETE statement that causes trigger activation.

For example, ordinarily it is illegal to INSERT rows in a view, but it is legal to create a trigger which intercepts attempts to INSERT, and puts rows in the underlying base table:

```
CREATE TABLE t1 (column1 INTEGER PRIMARY KEY, column2 INTEGER);
CREATE VIEW v1 AS SELECT column1, column2 FROM t1;
CREATE TRIGGER v1_instead_of INSTEAD OF INSERT ON v1 FOR EACH ROW BEGIN
 INSERT INTO t1 VALUES (NEW.column1, NEW.column2); END;
INSERT INTO v1 VALUES (1, 1);
-- ... The result will be: table t1 will contain a new row.
```

INSTEAD OF triggers are only legal for views, while BEFORE or AFTER triggers are only legal for base tables.

It is legal to create INSTEAD OF triggers with triggered WHEN clauses.

Limitations:

- It is legal to create INSTEAD OF triggers with UPDATE OF *column-list* clauses, but they are not standard SQL.

Пример:

```
CREATE TRIGGER ev1_instead_of_update
  INSTEAD OF UPDATE OF column2,column1 ON ev1
  FOR EACH ROW BEGIN
  INSERT INTO et2 VALUES (NEW.column1, NEW.column2); END;
```

### DROP TRIGGER

Syntax:

`DROP TRIGGER [IF EXISTS]` *`trigger-name`* `;`

Drop a trigger.

The *trigger-name* must identify a trigger that was created earlier with the *CREATE TRIGGER* statement.

Rules: none

Actions:

1. Tarantool returns an error if the trigger does not exist and there is no `IF EXISTS` clause.
2. The trigger is dropped.
3. Usually Tarantool effectively executes a *COMMIT* statement.

Примеры:

```
-- the simple case:
DROP TRIGGER table1_before_insert;
-- with an IF EXISTS clause:
DROP TRIGGER IF EXISTS table1_before_insert;
```

## Statements that change data

### INSERT

Syntax:

- `INSERT INTO` *`table-name`* `[(column-list)] VALUES (expression-list) [, (expression-list)];`
- `INSERT INTO` *`table-name`* `[(column-list)] select-statement;`

- `INSERT INTO table-name DEFAULT VALUES;`

Insert one or more new rows into a table.

The *table-name* must be a name of a table defined earlier with *CREATE TABLE*.

The optional *column-list* must be a comma-separated list of names of columns in the table.

The *expression-list* must be a comma-separated list of expressions; each expression may contain literals and operators and subqueries and function invocations.

Rules:

- The values in the *expression-list* are evaluated from left to right.
- The order of the values in the *expression-list* must correspond to the order of the columns in the table, or (if a *column-list* is specified) to the order of the columns in the *column-list*.
- The data type of the value should correspond to the *data type of the column*, that is, the data type that was specified with CREATE TABLE.
- If a *column-list* is not specified, then the number of expressions must be the same as the number of columns in the table.
- If a *column-list* is specified, then some columns may be omitted; omitted columns will get default values.
- The parenthesized *expression-list* may be repeated – `(expression-list),(expression-list),...` – for multiple rows.

Actions:

1. Tarantool evaluates each expression in *expression-list*, and returns an error if any of the rules is violated.
2. Tarantool creates zero or more new rows containing values based on the values in the VALUES list or based on the results of the *select-expression* or based on the default values.
3. Tarantool executes constraint checks and trigger actions and the actual insertion.

Примеры:

```
-- the simplest form:
INSERT INTO table1 VALUES (1, 'A');
-- with a column list:
INSERT INTO table1 (column1, column2) VALUES (2, 'B');
-- with an arithmetic operator in the first expression:
INSERT INTO table1 VALUES (2 + 1, 'C');
-- put two rows in the table:
INSERT INTO table1 VALUES (4, 'D'), (5, 'E');
```

See also: *REPLACE statement*.

### UPDATE

Syntax:

```
UPDATE table-name SET column-name = expression [, column-name = expression ...] [WHERE
search-condition];
```

Update zero or more existing rows in a table.

The *table-name* must be a name of a table defined earlier with *CREATE TABLE* or *CREATE VIEW*.

The *column-name* must be an updatable column in the table.

The *expression* may contain literals and operators and subqueries and function invocations and column names.

Rules:

- The values in the SET clause are evaluated from left to right.
- The data type of the value should correspond to the *data type of the column*, that is, the data type that was specified with CREATE TABLE.
- If a *search-condition* is not specified, then all rows in the table will be updated; otherwise only those rows which match the *search-condition* will be updated.

Actions:

1. Tarantool evaluates each expression in the SET clause, and returns an error if any of the rules is violated. For each row that is found by the WHERE clause, a temporary new row is formed based on the original contents and the modifications caused by the SET clause.
2. Tarantool executes constraint checks and trigger actions and the actual update.

Примеры:

```
-- the simplest form:
UPDATE t SET column1 = 1;
-- with more than one assignment in the SET clause:
UPDATE t SET column1 = 1, column2 = 2;
-- with a WHERE clause:
UPDATE t SET column1 = 5 WHERE column2 = 6;
```

Special cases:

It is legal to say SET (list of columns) = (list of values). For example:

```
UPDATE t SET (column1, column2, column3) = (1, 2, 3);
```

It is not legal to assign to a column more than once. For example:

```
INSERT INTO t (column1) VALUES (0);
UPDATE t SET column1 = column1 + 1, column1 = column1 + 1;
```

The result is an error: «duplicate column name».

It is not legal to assign to a primary-key column.

### DELETE

Syntax:

DELETE FROM *table-name* [WHERE search-condition];

Delete zero or more existing rows in a table.

The *table-name* must be a name of a table defined earlier with *CREATE TABLE* or *CREATE VIEW*.

The *search-condition* may contain literals and operators and subqueries and function invocations and column names.

Rules:

- If a search-condition is not specified, then all rows in the table will be deleted; otherwise only those rows which match the *search-condition* will be deleted.

Actions:

1. Tarantool evaluates each expression in the *search-condition*, and returns an error if any of the rules is violated.
2. Tarantool finds the set of rows that are to be deleted.
3. Tarantool executes constraint checks and trigger actions and the actual deletion.

Примеры:

```
-- the simplest form:
DELETE FROM t;
-- with a WHERE clause:
DELETE FROM t WHERE column2 = 6;
```

### REPLACE

Syntax:

- REPLACE INTO *table-name* [(column-list)] VALUES (expression-list) [, (expression-list)];
- REPLACE INTO *table-name* [(column-list)] select-statement;
- REPLACE INTO *table-name* DEFAULT VALUES;

Insert one or more new rows into a table, or update existing rows.

If a row already exists (as determined by the primary key or any unique key), then the action is delete + insert, and the rules are the same as for a *DELETE statement* followed by an *INSERT statement*. Otherwise the action is insert, and the rules are the same as for the INSERT statement.

Примеры:

```
-- the simplest form:
REPLACE INTO table1 VALUES (1, 'A');
-- with a column list:
REPLACE INTO table1 (column1, column2) VALUES (2, 'B');
-- with an arithmetic operator in the first expression:
REPLACE INTO table1 VALUES (2 + 1, 'C');
-- put two rows in the table:
REPLACE INTO table1 VALUES (4, 'D'), (5, 'E');
```

See also: *INSERT Statement*, *UPDATE Statement*.

### TRUNCATE

Syntax:

```
TRUNCATE TABLE table-name;
```

Remove all rows in the table.

TRUNCATE is considered to be a schema-change rather than a data-change statement, so it does not work within transactions (it cannot be *rolled back*).

Rules:

- It is illegal to truncate a table which is referenced by a foreign key.
- It is illegal to truncate a table which is also a system space, such as `_space`.
- The table must be a base table rather than a view.

Actions:

1. All rows in the table are removed. Usually this is faster than `DELETE FROM table-name;`.
2. If the table has an autoincrement primary key, its *sequence* is not reset to zero, but that may occur in a future Tarantool version.
3. There is no effect for any triggers associated with the table.
4. There is no effect on the counts for the `ROW_COUNT()` function.
5. Only one action is written to the *write-ahead log* (with `DELETE FROM table-name;` there would be one action for each deleted row).

Пример:

```
TRUNCATE TABLE t;
```

### SET

Syntax:

- `SET SESSION setting-name = setting-value;`

`SET SESSION` is a shorthand way to update the *box.space._session_settings* temporary system space.

`setting-name` can have the following values:

- `"sql_default_engine"`
- `"sql_full_column_names"`
- `"sql_full_metadata"`
- `"sql_parser_debug"`
- `"sql_recursive_triggers"`
- `"sql_reverse_unordered_selects"`
- `"sql_select_debug"`
- `"sql_vdbe_debug"`
- `"sql_defer_foreign_keys"` (removed in *2.11.0*)
- `"error_marshaling_enabled"` (removed in *2.10.0*)

The quote marks are necessary.

If `setting-name` is `"sql_default_engine"`, then `setting-value` can be either „vinyl“ or „memtx“. Otherwise, `setting-value` can be either TRUE or FALSE.

Example: `SET SESSION "sql_default_engine" = 'vinyl';` changes the default engine to „vinyl“ instead of „memtx“, and returns:

```
---
- row_count: 1
...
```

It is functionally the same thing as an *UPDATE Statement*:

```
UPDATE "_session_settings"
SET "value" = 'vinyl'
WHERE "name" = 'sql_default_engine';
```

## Statements that retrieve data

### SELECT

Syntax:

```
SELECT [ALL|DISTINCT] select list [from clause] [where clause] [group-by clause] [having
clause] [order-by clause];
```

Select zero or more rows.

The clauses of the SELECT statement are discussed in the following five sections.

### Select list

Syntax:

`select-list-column [, select-list-column ...]`

select-list-column:

Define what will be in a result set; this is a clause in a *SELECT statement*.

The *select list* is a comma-delimited list of expressions, or `*` (asterisk). An expression can have an alias provided with an `[[AS] column-name]` clause.

The `*` «asterisk» shorthand is valid if and only if the SELECT statement also contains a *FROM clause* which specifies the table or tables (details about the FROM clause are in the next section). The simple form is `*` which means «all columns» – for example, if the select is done for a table which contains three columns `s1` `s2` `s3`, then `SELECT * ...` is equivalent to `SELECT s1, s2, s3 ...`. The qualified form is `table-name.*` which means «all columns in the specified table», which again must be a result of the FROM clause – for example, if the table is named `table1`, then `table1.*` is equivalent to a list of the columns of `table1`.

The `[[AS] column-name]` clause determines the column name. The column name is useful for two reasons:

- in a tabular display, the column names are the headings
- if the results of the SELECT are used when creating a new table (such as a view), then the column names in the new table will be the column names in the *select list*.

If `[[AS] column-name]` is missing, and the expression is not simply the name of a column in the table, then Tarantool makes a name `COLUMN_`*n* where *n* is the number of the non-simple expression within the select list, for example `SELECT 5.88, table1.x, 'b' COLLATE "unicode_ci" FROM table1;` will cause the column names to be COLUMN_1, X, COLUMN_2. This is a behavior change since version *2.5.1*. In earlier versions, the name would be equal to the expression; see Issue#3962. It is still legal to define tables with column names like `COLUMN_1` but not recommended.

Примеры:

```
-- the simple form:
SELECT 5;
-- with multiple expressions including operators:
SELECT 1, 2 * 2, 'Three' || 'Four';
-- with [[AS] column-name] clause:
SELECT 5 AS column1;
-- * which must be eventually followed by a FROM clause:
SELECT * FROM table1;
-- as a list:
SELECT 1 AS a, 2 AS b, table1.* FROM table1;
```

### FROM clause

Syntax:

```
FROM [SEQSCAN] table-reference [, table-reference ...]
```

Specify the table or tables for the source of a *SELECT statement*.

The *table-reference* must be a name of an existing table, or a subquery, or a joined table.

A joined table looks like this:

```
table-reference-or-joined-table join-operator table-reference-or-joined-table
[join-specification]
```

A *join-operator* must be any of the standard types:

- [NATURAL] LEFT [OUTER] JOIN,
- [NATURAL] INNER JOIN, or
- CROSS JOIN

A *join-specification* must be any of:

- ON expression, or
- USING (column-name [, column-name ...])

Parentheses are allowed, and `[[AS] correlation-name]` is allowed.

The maximum number of joins in a FROM clause is 64.

The `SEQSCAN` keyword (since *2.11*) marks the queries that perform sequential scans during the execution. It happens if the query can't use indexes, and goes through all the table rows one by one, sometimes causing a heavy load. Such queries are called *scan queries*. If a scan query doesn't have the `SEQSCAN` keyword, Tarantool raises an error. `SEQSCAN` must precede all names of the tables that the query scans.

To find out if a query performs a sequential scan, use `EXPLAIN QUERY PLAN`. For scan queries, the result contains `SCAN TABLE table_name`.

---

**Примечание:** For backward compatibility, the scan queries without the `SEQSCAN` keyword are allowed in Tarantool 2.11. The errors on scan queries are the default behavior starting from 3.0. You can change the default behavior of scan queries using the *compat option sql_seq_scan*.

---

Примеры:

```
-- the simplest form:
SELECT * FROM SEQSCAN t;
-- with two tables, making a Cartesian join:
SELECT * FROM SEQSCAN t1, SEQSCAN t2;
-- with one table joined to itself, requiring correlation names:
SELECT a.*, b.* FROM SEQSCAN t1 AS a, SEQSCAN t1 AS b;
-- with a left outer join:
SELECT * FROM SEQSCAN t1 LEFT JOIN SEQSCAN t2;
```

### WHERE clause

Syntax:

```
WHERE condition;
```

Specify the condition for filtering rows from a table; this is a clause in a *SELECT* or *UPDATE* or *DELETE* statement.

The condition may contain any expression that returns a BOOLEAN (TRUE or FALSE or UNKNOWN) value.

For each row in the table:

- if the condition is true, then the row is kept;
- if the condition is false or unknown, then the row is ignored.

In effect, WHERE condition takes a table with n rows and returns a table with n or fewer rows.

Примеры:

```
-- with a simple condition:
SELECT 1 FROM t WHERE column1 = 5;
-- with a condition that contains AND and OR and parentheses:
SELECT 1 FROM t WHERE column1 = 5 AND (x > 1 OR y < 1);
```

### GROUP BY clause

Syntax:

```
GROUP BY expression [, expression ...]
```

Make a grouped table; this is a clause in a *SELECT statement*.

The expressions should be column names in the table, and each column should be specified only once.

In effect, the GROUP BY clause takes a table with rows that may have matching values, combines rows that have matching values into single rows, and returns a table which, because it is the result of GROUP BY, is called a grouped table.

Thus, if the input is a table:

```
a    b      c
-    -      -
1    'a'    'b
1    'b'    'b'
2    'a'    'b'
3    'a'    'b'
1    'b'    'b'
```

then `GROUP BY a, b` will produce a grouped table:

```
a    b      c
-    -      -
1    'a'    'b'
1    'b'    'b'
2    'a'    'b'
3    'a'    'b'
```

The rows where column `a` and column `b` have the same value have been merged; column `c` has been preserved but its value should not be depended on – if the rows were not all „b“, Tarantool could pick any value.

It is useful to envisage a grouped table as having hidden extra columns for the aggregation of the values, for example:

```
a    b      c    COUNT(a) SUM(a) MIN(c)
-    -      -    -------- ------ ------
1    'a'    'b'         2      2    'b'
1    'b'    'b'         1      1    'b'
2    'a'    'b'         1      2    'b'
     'a'    'b'         1      3    'b'
```

These extra columns are what *aggregate functions* are for.

Примеры:

```
-- with a single column:
SELECT 1 FROM t GROUP BY column1;
-- with two columns:
SELECT 1 FROM t GROUP BY column1, column2;
```

Limitations:

- `SELECT s1, s2 FROM t GROUP BY s1;` is legal.
- `SELECT s1 AS q FROM t GROUP BY q;` is legal.
- `SELECT s1 FROM t GROUP by 1;` is legal.

### Aggregate functions

Syntax:

`function-name (one or more expressions)`

Apply a built-in aggregate function to one or more expressions and return a scalar value.

Aggregate functions are only legal in certain clauses of a *SELECT statement* for grouped tables. (A table is a grouped table if a GROUP BY clause is present.) Also, if an aggregate function is used in a *select list* and the GROUP BY clause is omitted, then Tarantool assumes `SELECT ... GROUP BY [all columns];`.

NULLs are ignored for all aggregate functions except COUNT(*).

`AVG([DISTINCT] expression)` Return the average value of expression.

Example: `AVG(column1)`

`COUNT([DISTINCT] expression)` Return the number of occurrences of expression.

Example: `COUNT(column1)`

`COUNT(*)` Return the number of occurrences of a row.

Example: `COUNT(*)`

`GROUP_CONCAT(expression-1 [, expression-2])` **or** `GROUP_CONCAT(DISTINCT expression-1)` Return a list of *expression-1* values, separated by commas if *expression-2* is omitted, or separated by the *expression-2* value if *expression-2* is not omitted.

Example: `GROUP_CONCAT(`*`column1`*`)`

`MAX([DISTINCT] expression)` Return the maximum value of expression.

Example: `MAX(`*`column1`*`)`

`MIN([DISTINCT] expression)` Return the minimum value of expression.

Example: `MIN(`*`column1`*`)`

`SUM([DISTINCT] expression)` Возвращает сумму значений выражения или `NULL`, если строк не было.

Example: `SUM(`*`column1`*`)`

`TOTAL([DISTINCT] expression)` Возвращает сумму значений выражения или ноль, если строк не было.

Example: `TOTAL(`*`column1`*`)`

### HAVING clause

Syntax:

```
HAVING condition;
```

Specify the condition for filtering rows from a grouped table; this is a clause in a *SELECT statement*.

The clause preceding the HAVING clause may be a GROUP BY clause. HAVING operates on the table that the GROUP BY produces, which may contain grouped columns and aggregates.

If the preceding clause is not a GROUP BY clause, then there is only one group and the HAVING clause may only contain aggregate functions or literals.

For each row in the table:

- if the condition is true, then the row is kept;
- if the condition is false or unknown, then the row is ignored.

In effect, HAVING condition takes a table with n rows and returns a table with n or fewer rows.

Примеры:

```
-- with a simple condition:
SELECT 1 FROM t GROUP BY column1 HAVING column2 > 5;
-- with a more complicated condition:
SELECT 1 FROM t GROUP BY column1 HAVING column2 > 5 OR column2 < 5;
-- with an aggregate:
SELECT x, SUM(y) FROM t GROUP BY x HAVING SUM(y) > 0;
-- with no GROUP BY and an aggregate:
SELECT SUM(y) FROM t GROUP BY x HAVING MIN(y) < MAX(y);
```

Limitations:

- HAVING without GROUP BY is not supported for multiple tables.

### ORDER BY clause

Syntax:

```
ORDER BY expression [ASC|DESC] [, expression [ASC|DESC] ...]
```

Put rows in order; this is a clause in a *SELECT statement*.

An ORDER BY expression has one of three types which are checked in order:

1. Expression is a positive integer, representing the ordinal position of the column in the *select list*. For example, in the statement `SELECT x, y, z FROM t ORDER BY 2;` `ORDER BY 2` means «order by the second column in the select list», which is `y`.
2. Expression is a name of a column in the select list, which is determined by an AS clause. For example, in the statement `SELECT x, y AS x, z FROM t ORDER BY x;` `ORDER BY x` means «order by the column explicitly named `x` in the select list», which is the second column.
3. Expression contains a name of a column in a table of the FROM clause. For example, in the statement `SELECT x, y FROM t1 JOIN t2 ORDER BY z;` `ORDER BY z` means «order by a column named `z` which is expected to be in table `t1` or table `t2`».

If both tables contain a column named `z`, then Tarantool will choose the first column that it finds.

Выражение может содержать операторы и имена функций, а также конкретные значения. Например, в инструкции `SELECT x, y FROM t ORDER BY UPPER(z);` часть `ORDER BY UPPER(z)` означает «упорядочить по значениям столбца `t.z`, представленным в заглавном регистре». Вероятно, эта операция аналогична упорядочиванию с помощью одной из нечувствительных к регистру сортировок Tarantool.

Type 3 is illegal if the SELECT statement contains *UNION or EXCEPT or INTERSECT*.

If an ORDER BY clause contains multiple expressions, then expressions on the left are processed first and expressions on the right are processed only if necessary for tie-breaking. For example, in the statement `SELECT x, y FROM t ORDER BY x, y;` if there are two rows which both have the same values for column `x`, then an additional check is made to see which row has a greater value for column `y`.

In effect, ORDER BY clause takes a table with rows that may be out of order, and returns a table with rows in order.

Sorting order:

- The default order is ASC (ascending), the optional order is DESC (descending).
- Первыми идут значения NULL, затем значения типа BOOLEAN, числовые значения, STRING, VARBINARY и, наконец, UUID.
- Ordering does not matter for ARRAYs or MAPs or ANYs because they are not legal for comparisons.
- Within STRINGs, ordering is according to collation.
- Collation may be specified with a *COLLATE clause* within the ORDER BY column-list, or may be default.

Примеры:

```
-- with a single column:
SELECT 1 FROM t ORDER BY column1;
-- with two columns:
SELECT 1 FROM t ORDER BY column1, column2;
-- with a variety of data:
CREATE TABLE h (s1 NUMBER PRIMARY KEY, s2 SCALAR);
INSERT INTO h VALUES (7, 'A'), (4, 'a'), (-4, 'AZ'), (17, 17), (23, NULL);
INSERT INTO h VALUES (17.5, 'Д'), (1e+300, 'A'), (0, ''), (-1, '');
SELECT * FROM h ORDER BY s2 COLLATE "unicode_ci", s1;
-- The result of the above SELECT will be:
- - [23, null]
  - [17, 17]
  - [-1, '']
  - [0, '']
  - [4, 'a']
  - [7, 'A']
  - [1e+300, 'A']
  - [-4, 'AZ']
  - [17.5, 'Д']
...
```

Limitations:

- ORDER BY 1 is legal. This is common but is not standard SQL nowadays.

### LIMIT clause

Syntax:

- `LIMIT limit-expression [OFFSET offset-expression]`
- `LIMIT offset-expression, limit-expression`

---

**Примечание:** The above is not a typo: *offset-expression* and *limit-expression* are in reverse order if a comma is used.

---

Specify a maximum number of rows and a start row; this is a clause in a *SELECT statement*.

Expressions may contain integers and arithmetic operators or functions, for example `ABS(-3 / 1)`. However, the result must be an integer value greater than or equal to zero.

Usually the LIMIT clause follows an *ORDER BY clause*, because otherwise Tarantool does not guarantee that rows are in order.

Примеры:

```
-- simple case:
SELECT * FROM t LIMIT 3;
-- both limit and order:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
SELECT * FROM t LIMIT 3 OFFSET 1;
-- applied to a UNIONed result (LIMIT clause must be the final clause):
SELECT column1 FROM table1 UNION SELECT column1 FROM table2 ORDER BY 1 LIMIT 1;
```

Limitations:

- If ORDER BY ... LIMIT is used, then all order-by columns must be ASC or all must be DESC.

### Subquery

Syntax:

- *SELECT-statement* syntax
- *VALUES-statement* syntax

A subquery has the same syntax as a *SELECT statement* or *VALUES statement* embedded inside a main statement.

---

**Примечание:** The SELECT and VALUES statements are called «queries» because they return answers, in the form of result sets.

---

Subqueries may be the second part of *INSERT statements*. For example:

```
INSERT INTO t2 SELECT a, b, c FROM t1;
```

Subqueries may be in the *FROM clause* of SELECT statements.

Subqueries may be expressions, or be inside expressions. In this case they must be parenthesized, and usually the number of rows must be 1. For example:

```
SELECT 1, (SELECT 5), 3 FROM t WHERE c1 * (SELECT COUNT(*) FROM t2) > 5;
```

Subqueries may be expressions on the right side of certain comparison operators, and in this unusual case the number of rows may be greater than 1. The comparison operators are: [NOT] EXISTS and [NOT] IN. For example:

```
DELETE FROM t WHERE s1 NOT IN (SELECT s2 FROM t);
```

Subqueries may refer to values in the outer query. In this case, the subquery is called a «correlated subquery».

Subqueries may refer to rows which are being updated or deleted by the main query. In that case, the subquery finds the matching rows first, before starting to update or delete. For example, after:

```
CREATE TABLE t (s1 INTEGER PRIMARY KEY, s2 INTEGER);
INSERT INTO t VALUES (1, 3), (2, 1);
DELETE FROM t WHERE s2 NOT IN (SELECT s1 FROM t);
```

only one of the rows is deleted, not both rows.

### WITH clause

#### WITH clause (common table expression)

Syntax:

WITH *temporary-table-name* AS (subquery) [, *temporary-table-name* AS (subquery)] SELECT statement | INSERT statement | DELETE statement | UPDATE statement | REPLACE statement;

```
WITH v AS (SELECT * FROM t) SELECT * FROM v;
```

is equivalent to *creating a view* and selecting from it:

```
CREATE VIEW v AS SELECT * FROM t;
SELECT * FROM v;
```

The difference is that a WITH-clause «view» is temporary and only useful within the same statement. No CREATE privilege is required.

The WITH-clause can also be thought of as a *subquery* that has a name. This is useful when the same subquery is being repeated. For example:

```
SELECT * FROM t WHERE a < (SELECT s1 FROM x) AND b < (SELECT s1 FROM x);
```

can be replaced with:

```
WITH s AS (SELECT s1 FROM x) SELECT * FROM t,s WHERE a < s.s1 AND b < s.s1;
```

This «factoring out» of a repeated expression is regarded as good practice.

Примеры:

```
WITH cte AS (VALUES (7, '') INSERT INTO j SELECT * FROM cte;
WITH cte AS (SELECT s1 AS x FROM k) SELECT * FROM cte;
WITH cte AS (SELECT COUNT(*) FROM k WHERE s2 < 'x' GROUP BY s3)
  UPDATE j SET s2 = 5
  WHERE s1 = (SELECT s1 FROM cte) OR s3 = (SELECT s1 FROM cte);
```

WITH can only be used at the beginning of a statement, therefore it cannot be used at the beginning of a subquery or after a *set operator* or inside a CREATE statement.

A WITH-clause «view» is read-only because Tarantool does not support updatable views.

## WITH RECURSIVE

### WITH RECURSIVE clause (iterative common table expression)

The real power of WITH lies in the WITH RECURSIVE clause, which is useful when it is combined with *UNION or UNION ALL*:

```
WITH RECURSIVE recursive-table-name AS (SELECT ... FROM non-recursive-table-name ... UNION
[ALL] SELECT ... FROM recursive-table-name ...) statement-that-uses-recursive-table-name;
```

In non-SQL this can be read as: starting with a seed value from a non-recursive table, produce a recursive viewed table, UNION that with itself, UNION that with itself, UNION that with itself ... forever, or until a condition in the WHERE clause says «stop».

Пример:

```
CREATE TABLE ts (s1 INTEGER PRIMARY KEY);
INSERT INTO ts VALUES (1);
WITH RECURSIVE w AS (
  SELECT s1 FROM ts
  UNION ALL
  SELECT s1 + 1 FROM w WHERE s1 < 4)
SELECT * FROM w;
```

First, table `w` is seeded from `t1`, so it has one row: [1].

Then, `UNION ALL (SELECT s1 + 1 FROM w)` takes the row from `w` – which contains [1] – adds 1 because the select list says «s1+1», and so it has one row: [2].

Then, `UNION ALL (SELECT s1 + 1 FROM w)` takes the row from `w` – which contains [2] – adds 1 because the select list says «s1+1», and so it has one row: [3].

Then, `UNION ALL (SELECT s1 + 1 FROM w)` takes the row from `w` – which contains [3] – adds 1 because the select list says «s1+1», and so it has one row: [4].

Then, `UNION ALL (SELECT s1 + 1 FROM w)` takes the row from `w` – which contains [4] – and now the importance of the WHERE clause becomes evident, because «s1 < 4» is false for this row, and therefore the «stop» condition has been reached.

So, before the «stop», table `w` got 4 rows – [1], [2], [3], [4] – and the result of the statement looks like:

```
tarantool> WITH RECURSIVE w AS (
         >   SELECT s1 FROM ts
         >   UNION ALL
         >   SELECT s1 + 1 FROM w WHERE s1 < 4)
         > SELECT * FROM w;
---
- - [1]
  - [2]
  - [3]
  - [4]
...
```

In other words, this `WITH RECURSIVE ... SELECT` produces a table of auto-incrementing values.

### UNION, EXCEPT, and INTERSECT clauses

Syntax:

- `select-statement UNION [ALL] select-statement [ORDER BY clause] [LIMIT clause];`
- `select-statement EXCEPT select-statement [ORDER BY clause] [LIMIT clause];`
- `select-statement INTERSECT select-statement [ORDER BY clause] [LIMIT clause];`

UNION, EXCEPT, and INTERSECT are collectively called «set operators» or «table operators». In particular:

- `a UNION b` means «take rows which occur in a OR b».
- `a EXCEPT b` means «take rows which occur in a AND NOT b».
- `a INTERSECT b` means «take rows which occur in a AND b».

Duplicate rows are eliminated unless ALL is specified.

The *select-statements* may be chained: `SELECT ... SELECT ... SELECT ...;`

Each *select-statement* must result in the same number of columns.

The *select-statements* may be replaced with *VALUES statements*.

The maximum number of set operations is 50.

Пример:

```
CREATE TABLE t1 (s1 INTEGER PRIMARY KEY, s2 STRING);
CREATE TABLE t2 (s1 INTEGER PRIMARY KEY, s2 STRING);
INSERT INTO t1 VALUES (1, 'A'), (2, 'B'), (3, NULL);
INSERT INTO t2 VALUES (1, 'A'), (2, 'C'), (3,NULL);
SELECT s2 FROM t1 UNION SELECT s2 FROM t2;
SELECT s2 FROM t1 UNION ALL SELECT s2 FROM t2 ORDER BY s2;
SELECT s2 FROM t1 EXCEPT SELECT s2 FROM t2;
SELECT s2 FROM t1 INTERSECT SELECT s2 FROM t2;
```

В данном примере:

- The UNION query returns 4 rows: NULL, „A“, „B“, „C“.
- The UNION ALL query returns 6 rows: NULL, NULL, „A“, „A“, „B“, „C“.
- The EXCEPT query returns 1 row: „B“.
- The INTERSECT query returns 2 rows: NULL, „A“.

Limitations:

- Parentheses are not allowed.

- Evaluation is left to right, INTERSECT does not have precedence.

Пример:

```
CREATE TABLE t01 (s1 INTEGER PRIMARY KEY, s2 STRING);
CREATE TABLE t02 (s1 INTEGER PRIMARY KEY, s2 STRING);
CREATE TABLE t03 (s1 INTEGER PRIMARY KEY, s2 STRING);
INSERT INTO t01 VALUES (1, 'A');
INSERT INTO t02 VALUES (1, 'B');
INSERT INTO t03 VALUES (1, 'A');
SELECT s2 FROM t01 INTERSECT SELECT s2 FROM t03 UNION SELECT s2 FROM t02;
SELECT s2 FROM t03 UNION SELECT s2 FROM t02 INTERSECT SELECT s2 FROM t03;
-- ... results are different.
```

### INDEXED BY clause

Syntax:

INDEXED BY *index-name*

The INDEXED BY clause may be used in a *SELECT*, *DELETE*, or *UPDATE* statement, immediately after the *table-name*. For example:

```
DELETE FROM table7 INDEXED BY index7 WHERE column1 = 'a';
```

In this case the search for „a“ will take place within `index7`. For example:

```
SELECT * FROM table7 NOT INDEXED WHERE column1 = 'a';
```

In this case the search for „a“ will be done via a search of the whole table, what is sometimes called a «full table scan», even if there is an index for `column1`.

Ordinarily Tarantool chooses the appropriate index or lookup method depending on a complex set of «optimizer» rules; the INDEXED BY clause overrides the optimizer choice. If the index was defined with the *exclude_ null* parts option, it will only be used if the user specifies it.

Пример:

Suppose a table has two columns:

- The first column is the primary key and therefore it has an automatic index named `pk_unnamed_T_1`.
- The second column has an index created by the user.

The user selects with `INDEXED BY the-index-on-column1`, then selects with `INDEXED BY the-index-on-column-2`.

```
CREATE TABLE t (column1 INTEGER PRIMARY KEY, column2 INTEGER);
CREATE INDEX idx_column2_t_1 ON t (column2);
INSERT INTO t VALUES (1, 2), (2, 1);
SELECT * FROM t INDEXED BY "pk_unnamed_T_1";
SELECT * FROM t INDEXED BY idx_column2_t_1;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Result for the first select: (1, 2), (2, 1)
-- Result for the second select: (2, 1), (1, 2).
```

Limitations: Often INDEXED BY has no effect. Often INDEXED BY affects a choice of covering index, but not a WHERE clause.

### VALUES

Syntax:

```
VALUES (expression [, expression ...]) [, (expression [, expression ...])
```

Select one or more rows.

VALUES has the same effect as *SELECT*, that is, it returns a result set, but VALUES statements may not have FROM or GROUP or ORDER BY or LIMIT clauses.

VALUES may be used wherever SELECT may be used, for example in *subqueries*.

Примеры:

```
-- simple case:
VALUES (1);
-- equivalent to SELECT 1, 2, 3:
VALUES (1, 2, 3);
-- two rows:
VALUES (1, 2, 3), (4, 5, 6);
```

### PRAGMA

Syntax:

- `PRAGMA` *`pragma-name`* `(pragma-value);`
- or `PRAGMA` *`pragma-name`* `;`

PRAGMA statements will give rudimentary information about database „metadata“ or server performance, although it is better to get metadata via *system tables*.

For PRAGMA statements that include (`pragma-value`), pragma values are strings and can be specified inside `""` double quotes, or without quotes. When a string is used for searching, results must match according to a binary collation. If the object being searched has a lower-case name, use double quotes.

In an earlier version, there were some PRAGMA statements that determined behavior. Now that does not happen. Behavior change is done by updating the *box.space._session_settings* system table.

| Pragma | Parameter | Эффект |
|---|---|---|
| foreign_key_list | string table-name | Return a *result set* with one row for each foreign key of «table-name». Each row contains: (INTEGER) id – identification number (INTEGER) seq – sequential number (STRING) table – name of table (STRING) from – referencing key (STRING) to – referenced key (STRING) on_update – ON UPDATE clause (STRING) on_delete – ON DELETE clause (STRING) match – MATCH clause The system table is `"_fk_constraint"`. |
| collation_list | | Return a result set with one row for each supported collation. The first four collations are `'none'` and `'unicode'` and `'unicode_ci'` and `'binary'`, then come about 270 predefined collations, the exact count may vary because users can add their own collations. The system table is `"_collation"`. |
| index_info | string table-name . index-name | Return a result set with one row for each column in «table-name.index-name». Each row contains: (INTEGER) seqno – the column's ordinal position in the index (first column is 0) (INTEGER) cid – the column's ordinal position in the table (first column is 0) (STRING) name – name of the column (INTEGER) desc – 0 is ASC, 1 is DESC (STRING) collation name (STRING) type – data type |
| index_list | string table-name | Return a result set with one row for each index of «table-name». Each row contains: (INTEGER) seq – sequential number (STRING) name – index name (INTEGER) unique – whether the index is unique, 0 is false, 1 is true The system table is `"_index"`. |
| stats | | Return a result set with one row for each index of each table. Each row contains: (STRING) table – name of the table (STRING) index – name of the index (INTEGER) width – arbitrary information (INTEGER) height – arbitrary information |
| table_info | string table-name | Return a result set with one row for each column in «table-name». Each row contains: (INTEGER) cid – ordinal position in the table (first column number is 0) (STRING) name – column name (STRING) type (INTEGER) notnull – whether the column is NOT NULL, 0 is false, 1 is true. (STRING) dflt_value – default value (INTEGER) pk – whether the column is a PRIMARY KEY column, 0 is false, 1 is true. |

Example: (not showing result set metadata)

```
PRAGMA table_info(T);
---
- - [0, 's1', 'integer', 1, null, 1]
  - [1, 's2', 'integer', 0, null, 0]
...
```

## EXPLAIN

Syntax:

- `EXPLAIN explainable-statement;`

EXPLAIN will show what steps Tarantool would take if it executed explainable-statement. This is primarily a debugging and optimization aid for the Tarantool team.

Example: `EXPLAIN DELETE FROM m;` returns:

```
- - [0, 'Init', 0, 3, 0, '', '00', 'Start at 3']
  - [1, 'Clear', 16416, 0, 0, '', '00', '']
  - [2, 'Halt', 0, 0, 0, '', '00', '']
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  - [3, 'Transaction', 0, 1, 1, '0', '01', 'usesStmtJournal=0']
  - [4, 'Goto', 0, 1, 0, '', '00', '']
```

Variation: `EXPLAIN QUERY PLAN statement;` shows the steps of a search.

### Statements for transactions

## START TRANSACTION

Syntax:

```
START TRANSACTION;
```

Start a transaction. After `START TRANSACTION;`, a transaction is «active». If a transaction is already active, then `START TRANSACTION;` is illegal.

Transactions should be active for fairly short periods of time, to avoid concurrency issues. To end a transaction, say *COMMIT;* or *ROLLBACK;*.

Just as in NoSQL, transaction control statements are subject to limitations set by the *storage engine* involved:
* For the memtx storage engine, if a yield happens within an active transaction, the transaction is rolled back.
* For the vinyl engine, yields are allowed. Also, although CREATE AND DROP and ALTER statements are legal in transactions, there are a few exceptions. For example, `CREATE INDEX ON` *`table_name`* `...` will fail within a multi-statement transaction if the table is not empty.

Однако инструкции управления транзакциями всё еще могут работать не так, как вы ожидаете при запуске через сетевое соединение: транзакция связана с файбером, а не с сетевым соединением, и разные инструкции управления транзакциями, отправленные через одно и то же сетевое соединение, могут быть выполнены разными файберами из пула файберов.

In order to ensure that all statements are part of the intended transaction, put all of them between `START TRANSACTION;` and `COMMIT;` or `ROLLBACK;` then send as a single batch. For example:

- Enclose each separate SQL statement in a *box.execute()* function.
- Pass all the `box.execute()` functions to the server in a single message.

  If you are using a console, you can do this by writing everything on a single line.

  If you are using *net.box*, you can do this by putting all the function calls in a single string and calling *eval(string)*.

Пример:

```
START TRANSACTION;
```

Example of a whole transaction sent to a server on `localhost:3301` with `eval(string)`:

```
net_box = require('net.box')
conn = net_box.new('localhost', 3301)
s = 'box.execute([[START TRANSACTION;]]) '
s = s .. 'box.execute([[INSERT INTO t VALUES (1);]]) '
```

(continues on next page)

(продолжение с предыдущей страницы)

```
s = s .. 'box.execute([[ROLLBACK;]]) '
conn:eval(s)
```

### COMMIT

Syntax:

```
COMMIT;
```

Commit an active transaction, so all changes are made permanent and the transaction ends.

COMMIT is illegal unless a transaction is active. If a transaction is not active then SQL statements are committed automatically.

Пример:

```
COMMIT;
```

### SAVEPOINT

Syntax:

```
SAVEPOINT savepoint-name;
```

Set a savepoint, so that *ROLLBACK TO savepoint-name* is possible.

SAVEPOINT is illegal unless a transaction is active.

If a savepoint with the same name already exists, it is released before the new savepoint is set.

Пример:

```
SAVEPOINT x;
```

### RELEASE SAVEPOINT

Syntax:

```
RELEASE SAVEPOINT savepoint-name;
```

Release (destroy) a savepoint created by a *SAVEPOINT statement*.

RELEASE is illegal unless a transaction is active.

Savepoints are released automatically when a transaction ends.

Пример:

```
RELEASE SAVEPOINT x;
```

### ROLLBACK

Syntax:

ROLLBACK [TO [SAVEPOINT] *savepoint-name*];

If ROLLBACK does not specify a *savepoint-name*, rollback an active transaction, so all changes since *START TRANSACTION* are cancelled, and the transaction ends.

If ROLLBACK does specify a *savepoint-name*, rollback an active transaction, so all changes since *SAVEPOINT savepoint-name* are cancelled, and the transaction does not end.

ROLLBACK is illegal unless a transaction is active.

Примеры:

```
-- the simple form:
ROLLBACK;
-- the form so changes before a savepoint are not cancelled:
ROLLBACK TO SAVEPOINT x;
```

```
-- An example of a Lua function that will do a transaction
-- containing savepoint and rollback to savepoint.
function f()
box.execute([[DROP TABLE IF EXISTS t;]]) -- commits automatically
box.execute([[CREATE TABLE t (s1 STRING PRIMARY KEY);]]) -- commits automatically
box.execute([[START TRANSACTION;]]) -- after this succeeds, a transaction is active
box.execute([[INSERT INTO t VALUES ('Data change #1');]])
box.execute([[SAVEPOINT "1";]])
box.execute([[INSERT INTO t VALUES ('Data change #2');]])
box.execute([[ROLLBACK TO SAVEPOINT "1";]]) -- rollback Data change #2
box.execute([[ROLLBACK TO SAVEPOINT "1";]]) -- this is legal but does nothing
box.execute([[COMMIT;]]) -- make Data change #1 permanent, end the transaction
end
```

### Functions

### Explanation of functions

Syntax:

`function-name (one or more expressions)`

Apply a built-in function to one or more expressions and return a scalar value.

Tarantool поддерживает 33 встроенные функции.

The maximum number of operands for any function is 127.

The required privileges for built-in functions will likely change in a future version.

### List of functions

Ниже приведены встроенные функции Tarantool/SQL. Начиная с Tarantool 2.10 для функций, требующих числовые аргументы, аргументы с типом данных NUMBER недопустимы.

### ABS

Syntax:

`ABS(`*`numeric-expression`*`)`

Return the absolute value of numeric-expression, which can be any numeric type.

Example: `ABS(-1)` is 1.

### CAST

Syntax:

`CAST(`*`expression`* `AS` *`data-type`*`)`

Return the expression value after casting to the specified *data type*.

CAST в или из UUID может поменять порядок байтов на little-endian или обратно.

Examples: `CAST('AB' AS VARBINARY)`, `CAST(X'4142' AS STRING)`

### CHAR

Syntax:

`CHAR([numeric-expression [,numeric-expression...])`

Return the characters whose Unicode code point values are equal to the numeric expressions.

Short example:

The first 128 Unicode characters are the «ASCII» characters, so CHAR(65, 66, 67) is „ABC“.

Long example:

For the current list of Unicode characters, in order by code point, see www.unicode.org/Public/UCD/latest/ucd/UnicodeData.txt. In that list, there is a line for a Linear B ideogram

```
100CC;LINEAR B IDEOGRAM B240 WHEELED CHARIOT ...
```

Therefore, for a string with a chariot in the middle, use the concatenation operator `||` and the CHAR function

`'start of string ' || CHAR(0X100CC) || ' end of string'`.

### COALESCE

Syntax:

```
COALESCE(expression, expression [, expression ...])
```

Return the value of the first non-NULL expression, or, if all expression values are NULL, return NULL.

**Пример:** `COALESCE(NULL, 17, 32)` is 17.

### DATE_PART

Syntax:

```
DATE_PART(value_requested , datetime)
```

Since *2.10.0*.

The `DATE_PART()` function returns the requested information from a DATETIME value. It takes two arguments: the first one tells us what information is requested, the second is a DATETIME value.

Below is a list of supported values of the first argument and what information is returned:

- `millennium` – millennium
- `century` – century
- `decade` – decade
- `year` – year
- `quarter` – quarter of year
- `month` – month of year
- `week` – week of year
- `day` – day of month
- `dow` – day of week
- `doy` – day of year
- `hour` – hour of day
- `minute` – minute of hour
- `second` – second of minute
- `millisecond` – millisecond of second
- `microsecond` – microsecond of second
- `nanosecond` – nanosecond of second

- `epoch` – epoch
- `timezone_offset` – time zone offset from the UTC, in minutes.

Примеры:

```
tarantool> select date_part('millennium', cast({'year': 2000, 'month': 4, 'day': 5, 'hour
↪': 6, 'min': 33, 'sec': 22, 'nsec': 523999111} as datetime));
---
- metadata:
  - name: COLUMN_1
    type: integer
  rows:
  - [2]
...

tarantool> select date_part('day', cast({'year': 2000, 'month': 4, 'day': 5, 'hour': 6,
↪'min': 33, 'sec': 22, 'nsec': 523999111} as datetime));
---
- metadata:
  - name: COLUMN_1
    type: integer
  rows:
  - [5]
...

tarantool> select date_part('nanosecond', cast({'year': 2000, 'month': 4, 'day': 5, 'hour
↪': 6, 'min': 33, 'sec': 22, 'nsec': 523999111} as datetime));
---
- metadata:
  - name: COLUMN_1
    type: integer
  rows:
  - [523999111]
...
```

### GREATEST

Syntax:

`GREATEST(`*`expression-1`*`, `*`expression-2`*`, [`*`expression-3`*` ...])`

Return the greatest value of the supplied expressions, or, if any expression is NULL, return NULL. The reverse of `GREATEST` is *LEAST*.

Examples: `GREATEST(7, 44, -1)` is 44; `GREATEST(1E308, 'a', 0, X'00')` is „0“ = the nul character; `GREATEST(3, NULL, 2)` is NULL

### HEX

Syntax:

`HEX(expression)`

Возвращает шестнадцатеричный код для каждого байта выражения **expression**.

Начиная с версии Tarantool 2.10.0 выражение должно быть последовательностью байтов (тип данных VARBINARY).

В предыдущих версиях Tarantool выражение могло быть либо строкой, либо последовательностью байтов. Для символов ASCII оно выполнялось довольно прямолинейно, поскольку значение каждого символа в кодировке ASCII совпадает с его значением в шестнадцатеричной схеме. В случае с символами, не входящими в таблицу ASCII, для каждого символа потребуется два байта или более, поскольку строки символов обычно кодируются в UTF-8.

Примеры:

- `HEX(X'41')` вернет `41`.
- `HEX(CAST('Д' AS VARBINARY))` вернет `D094`.

### IFNULL

Syntax:

`IFNULL(expression, expression)`

Return the value of the first non-NULL expression, or, if both expression values are NULL, return NULL. Thus `IFNULL(expression, expression)` is the same as *COALESCE(expression, expression)*.

**Пример:** `IFNULL(NULL, 17)` is 17

### LEAST

Syntax:

`LEAST(`*`expression-1`*`, `*`expression-2`*`, [`*`expression-3`*` ...])`

Return the least value of the supplied expressions, or, if any expression is NULL, return NULL. The reverse of `LEAST` is *GREATEST*.

Examples: `LEAST(7, 44, -1)` is -1; `LEAST(1E308, 'a', 0, X'00')` is 0; `LEAST(3, NULL, 2)` is NULL.

### LENGTH

Syntax:

`LENGTH(expression)`

Return the number of characters in the **expression**, or the number of bytes in the **expression**. It depends on the data type: strings with data type STRING are counted in characters, byte sequences with data type VARBINARY are counted in bytes and are not ended by the nul character. There are two aliases for `LENGTH(expression)` – `CHAR_LENGTH(expression)` and `CHARACTER_LENGTH(expression)` do the same thing.

Примеры:

- `LENGTH('ДД')` is 2, the string has 2 characters.
- `LENGTH(CAST('ДД' AS VARBINARY))` is 4, the string has 4 bytes.
- `LENGTH(CHAR(0, 65))` is 2, „0“ does not mean „end of string“.
- `LENGTH(X'410041')` is 3, X“...“ byte sequences have type VARBINARY.

### LIKELIHOOD

Syntax:

`LIKELIHOOD(`*`expression`*`, `*`DOUBLE literal`*`)`

Возвращает выражение без изменений, при условии, что числовое значение находится в диапазоне от 0.0 до 1.0.

Example: `LIKELIHOOD('a' = 'b', .0)` is FALSE

### LIKELY

Syntax:

`LIKELY(`*`expression`*`)`

Return TRUE if the expression is probably true.

Example: `LIKELY('a' <= 'b')` is TRUE

### LOWER

Syntax:

`LOWER(`*`string-expression`*`)`

Return the expression, with upper-case characters converted to lower case. The reverse of `LOWER` is *UPPER*.

Example: `LOWER('ДА')` is „да“

### NOW

Syntax:

`NOW()`

Since *2.10.0*.

The NOW() function returns the current date and time as a DATETIME value.

If the function is called more than once in a query, it returns the same result until the query completes, unless a yield has occurred. On yield, the value returned by NOW() is changing.

Примеры:

```
tarantool> select now(), now(), now()
---
- metadata:
  - name: COLUMN_1
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    type: datetime
  - name: COLUMN_2
    type: datetime
  - name: COLUMN_3
    type: datetime
  rows:
  - ['2022-07-20T19:02:02.010812282+0300', '2022-07-20T19:02:02.010812282+0300', '2022-
↪07-20T19:02:02.010812282+0300']
...
```

### NULLIF

Syntax:

`NULLIF(expression-1, expression-2)`

Return *expression-1* if *expression-1* <> *expression-2*, otherwise return NULL.

Примеры:

- `NULLIF('a', 'A')` is „a“.
- `NULLIF(1.00, 1)` is NULL.

---

**Примечание:** Before *Tarantool 2.10.4*, the type of the result was always `SCALAR`. Since *Tarantool 2.10.4*, the result of `NULLIF` matches the type of the first argument. If the first argument is the `NULL` literal, then the result has the `SCALAR` type.

---

### POSITION

Syntax:

`POSITION(`*`expression-1`*`, `*`expression-2`*`)`

Return the position of expression-1 within expression-2, or return 0 if expression-1 does not appear within expression-2. The data types of the expressions must be either STRING or VARBINARY. If the expressions have data type STRING, then the result is the character position. If the expressions have data type VARBINARY, then the result is the byte position.

Short example: `POSITION('C', 'ABC')` is 3

Long example: The UTF-8 encoding for the Latin letter A is hexadecimal 41; the UTF-8 encoding for the Cyrillic letter Д is hexadecimal D094 – you can confirm this by saying SELECT HEX(„ДА“); and seeing that the result is „D09441“. If you now execute `SELECT POSITION('A', 'ДА');` the result will be 2, because „A“ is the second character in the string. However, if you now execute `SELECT POSITION(X'41', X'D09441');` the result will be 3, because X'41“ is the third byte in the byte sequence.

### PRINTF

Syntax:

`PRINTF(string-expression [, expression ...])`

Возвращает строку, отформатированную по правилам функции C `sprintf()`, где `%d%s` означает, что следующие два аргумента — числовой и строка, и так далее.

If an argument is missing or is NULL, it becomes:

- „0“ if the format requires an integer,
- `'0.0'`, если формат требует числового значения с десятичной точкой,
- „“ if the format requires a string.

Example: `PRINTF('%da', 5)` is „5a“.

### QUOTE

Syntax:

`QUOTE(string)`

Return a string with enclosing quotes if necessary, and with quotes inside the enclosing quotes if necessary. This function is useful for creating strings which are part of SQL statements, because of SQL's rules that string literals are enclosed by single quotes, and single quotes inside such strings are shown as two single quotes in a row.

Начиная с версии Tarantool 2.10 аргументы с числовыми типами данных возвращаются без изменений.

Примеры: `QUOTE('a')` вернет `'a'`, `QUOTE(5)` вернет `5`.

### RAISE

Syntax:

`RAISE(FAIL, error-message)`

This may only be used within a triggered statement. See also *Trigger Activation*.

### RANDOM

Syntax: `RANDOM()`

Return a 19-digit integer which is generated by a pseudo-random number generator,

Example: `RANDOM()` is 6832175749978026034, or it is any other integer

### RANDOMBLOB

Syntax:

`RANDOMBLOB(`*n*`)`

Return a byte sequence, n bytes long, data type = VARBINARY, containing bytes generated by a pseudo-random byte generator. The result can be translated to hexadecimal. If n is less than 1 or is NULL or is infinity, then NULL is returned.

Example: `HEX(RANDOMBLOB(3))` is „9EAAA8“, or it is the hex value for any other three-byte string

### REPLACE

Syntax:

`REPLACE(`*expression-1*`, `*expression-2*`, `*expression-3*`)`

Return expression-1, except that wherever expression-1 contains expression-2, replace expression-2 with expression-3. The expressions should all have data type STRING or VARBINARY.

Example: `REPLACE('AAABCCCBD', 'B', '!')` is „AAA!CCC!D“

### ROUND

Syntax:

`ROUND(`*numeric-expression-1* `[, `*numeric-expression-2*`])`

Возвращает округленное значение первого числового выражения. Положительные числа округляются до 0,5 в большую сторону, отрицательные — до 0,5 в меньшую. Если задано второе числовое выражение, то первое округляется до ближайших цифр после десятичной точки второго выражения. Если второе выражение не задано, первое округляется до ближайшего целого числа.

Example: `ROUND(-1.5)` is -2, `ROUND(1.7766E1,2)` is 17.77.

### ROW_COUNT

`ROW_COUNT()`

Return the number of rows that were inserted / updated / deleted by the last *INSERT* or *UPDATE* or *DELETE* or *REPLACE* statement. Rows which were updated by an UPDATE statement are counted even if there was no change. Rows which were inserted / updated / deleted due to foreign-key action are not counted. Rows which were inserted / updated / deleted due to a view's *INSTEAD OF triggers* are not counted. After a CREATE or DROP statement, ROW_COUNT() is 1. After other statements, ROW_COUNT() is 0.

Example: `ROW_COUNT()` is 1 after a successful INSERT of a single row.

Special rule if there are BEFORE or AFTER triggers: In effect the ROW_COUNT() counter is pushed at the beginning of a series of triggered statements, and popped at the end. Therefore, after the following statements:

```
CREATE TABLE t1 (s1 INTEGER PRIMARY KEY);
CREATE TABLE t2 (s1 INTEGER, s2 STRING, s3 INTEGER, PRIMARY KEY (s1, s2, s3));
CREATE TRIGGER tt1 BEFORE DELETE ON t1 FOR EACH ROW BEGIN
  INSERT INTO t2 VALUES (old.s1, '#2 Triggered', ROW_COUNT());
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  INSERT INTO t2 VALUES (old.s1, '#3 Triggered', ROW_COUNT());
  END;
INSERT INTO t1 VALUES (1),(2),(3);
DELETE FROM t1;
INSERT INTO t2 VALUES (4, '#4 Untriggered', ROW_COUNT());
SELECT * FROM t2;
```

The result is:

```
---
- - [1, '#2 Triggered', 3]
  - [1, '#3 Triggered', 1]
  - [2, '#2 Triggered', 3]
  - [2, '#3 Triggered', 1]
  - [3, '#2 Triggered', 3]
  - [3, '#3 Triggered', 1]
  - [4, '#4 Untriggered', 3]
...
```

### SOUNDEX

Syntax:

`SOUNDEX(string-expression)`

Return a four-character string which represents the sound of `string-expression`. Often words and names which have different spellings will have the same Soundex representation if they are pronounced similarly, so it is possible to search by what they sound like. The algorithm works with characters in the Latin alphabet and works best with English words.

Example: `SOUNDEX('Crater')` and `SOUNDEX('Creature')` both return `C636`.

### SUBSTR

Syntax:

`SUBSTR(`*`string-or-varbinary-value`*`, `*`numeric-start-position`*` [, `*`numeric-length`*`])`

If string-or-varbinary-value has data type STRING, then return the substring which begins at character position numeric-start-position and continues for numeric-length characters (if numeric-length is supplied), or continues till the end of string-or-varbinary-value (if numeric-length is not supplied).

If numeric-start-position is less than 1, or if numeric-start-position + numeric-length is greater than the length of string-or-varbinary-value, then the result is not an error, anything which would be before the start or after the end is ignored. There are no symbols with index <= 0 or with index greater than the length of the first argument.

If numeric-length is less than 0, then the result is an error.

If string-or-varbinary-value has data type VARBINARY rather than data type STRING, then positioning and counting is by bytes rather than by characters.

Examples: `SUBSTR('ABCDEF', 3, 2)` is „CD“, `SUBSTR('абвгде', -1, 4)` is „аб“

### TRIM

Syntax:

`TRIM([[LEADING|TRAILING|BOTH] [`*expression-1*`] FROM] `*expression-2*`)`

Return expression-2 after removing all leading and/or trailing characters or bytes. The expressions should have data type STRING or VARBINARY. If LEADING|TRAILING|BOTH is omitted, the default is BOTH. If expression-1 is omitted, the default is „ „ (space) for data type STRING or X'00" (nul) for data type VARBINARY.

Примеры:

`TRIM('a' FROM 'abaaaaa')` is „b" – all repetitions of „a" are removed on both sides; `TRIM(TRAILING '' FROM '')` is „" – if all characters are Hebrew, TRAILING means «left»; `TRIM(X'004400')` is X'44" – the default byte sequence to trim is X'00" when data type is VARBINARY; `TRIM(LEADING 'abc' FROM 'abcd')` is „d" – expression-1 can have more than 1 character.

### TYPEOF

Syntax:

`TYPEOF(`*expression*`)`

Возвращает `'NULL'`, если выражение равно `NULL`. Возвращает `'scalar'`, если выражение — это имя колонки, определенной как SCALAR. В остальных случаях возвращает *тип данных* выражения.

Примеры:

`TYPEOF('A')` вернет `'string'`; `TYPEOF(RANDOMBLOB(1))` вернет `'varbinary'`; `TYPEOF(1e44)` вернет `'double'` или `'number'`; `TYPEOF(-44)` вернет `'integer'`; `TYPEOF(NULL)` вернет `'NULL'`.

До версии Tarantool 2.10 `TYPEOF(выражение)` возвращало тип данных результата выражения во всех случаях.

### UNICODE

Syntax:

`UNICODE(string-expression)`

Return the Unicode code point value of the first character of **string-expression**. If *string-expression* is empty, the return is NULL. This is the reverse of *CHAR(integer)*.

Example: `UNICODE('Щ')` is 1065 (hexadecimal 0429).

### UNLIKELY

Syntax:

`UNLIKELY(`*expression*`)`

Return TRUE if the expression is probably false. Limitation: in fact `UNLIKELY` may return the same thing as *LIKELY*.

Example: `UNLIKELY('a' <= 'b')` is TRUE.

### UPPER

Syntax:

`UPPER(string-expression)`

Return the expression, with lower-case characters converted to upper case. The reverse of `UPPER` is *LOWER*.

Example: `UPPER('-4щl')` is „-4ЩL“.

### UUID

Syntax:

`UUID([integer])`

Возвращает универсальный уникальный идентификатор, тип данных UUID. Опционально можно указать номер версии, однако на данный момент единственной возможной версией является 4, которая установлена по умолчанию. Поддержка UUID в SQL была добавлена в версии Tarantool 2.9.1.

Пример: `UUID()` или `UUID(4)`

### VERSION

Syntax:

`VERSION()`

Return the Tarantool version.

Example: for a February 2020 build VERSION() is `'2.4.0-35-g57f6fc932'`.

### ZEROBLOB

Syntax:

`ZEROBLOB(n)`

Return a byte sequence, data type = VARBINARY, n bytes long.

### COLLATE clause

`COLLATE collation-name`

The collation-name must identify an existing collation.

The COLLATE clause is allowed for STRING or SCALAR items: () in *CREATE INDEX* () in *CREATE TABLE* as part of *column definition* () in CREATE TABLE as part of *UNIQUE definition* () in string expressions

Примеры:

```
-- In CREATE INDEX
CREATE INDEX idx_unicode_mb_1 ON mb (s1 COLLATE "unicode");
-- In CREATE TABLE
CREATE TABLE t1 (s1 INTEGER PRIMARY KEY, s2 STRING COLLATE "unicode_ci");
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- In CREATE TABLE ... UNIQUE
CREATE TABLE mb (a STRING, b STRING, PRIMARY KEY(a), UNIQUE(b COLLATE "unicode_ci"␣
↪DESC));
-- In string expressions
SELECT 'a' = 'b' COLLATE "unicode"
    FROM t
    WHERE s1 = 'b' COLLATE "unicode"
    ORDER BY s1 COLLATE "unicode";
```

The list of collations can be seen with: *PRAGMA collation_list;*

The collation rules comply completely with the Unicode Technical Standard #10 («Unicode Collation Algorithm») and the default character order is as in the Default Unicode Collation Element Table (DUCET). There are many permanent collations; the commonly used ones include: `"none"` (not applicable) `"unicode"` (characters are in DUCET order with strength = „tertiary“) `"unicode_ci"` (characters are in DUCET order with strength = „primary“) `"binary"` (characters are in code point order) These identifiers must be quoted and in lower case because they are in lower case in *Tarantool/NoSQL collations*.

If one says `COLLATE "binary"`, this is equivalent to asking for what is sometimes called «code point order» because, if the contents are in the UTF-8 character set, characters with larger code points will appear after characters with lower code points.

In an expression, `COLLATE` is an operator with higher precedence than anything except `~`. This is fine because there are no other useful operators except `||` and comparison. After `||`, collation is preserved.

In an expression with more than one `COLLATE` clause, if the collation names differ, there is an error: «Illegal mix of collations». In an expression with no `COLLATE` clauses, literals have collation `"binary"`, columns have the collation specified by `CREATE TABLE`.

In other words, to pick a collation, Tarantool uses: the first `COLLATE` clause in an expression if it was specified, else the column's `COLLATE` clause if it was specified, else `"binary"`.

However, for searches and sometimes for sorting, the collation may be an index's collation, so all non-index `COLLATE` clauses are ignored.

*EXPLAIN* will not show the name of what collation was used, but will show the collation's characteristics.

Example with Swedish collation: Knowing that «sv» is the two-letter code for Swedish, and knowing that «s1» means strength = 1, and seeing with `PRAGMA collation_list;` that there is a collation named unicode_sv_s1, check whether two strings are equal according to Swedish rules (yes they are): `SELECT 'ÄÄ' = 'ĘĘ' COLLATE "unicode_sv_s1";`

Example with Russian and Ukrainian and Kyrgyz collations: Knowing that Russian collation is practically the same as Unicode default, and knowing that the two-letter codes for Ukrainian and Kyrgyz are „uk“ and „ky“, and knowing that in Russian (but not Ukrainian) „Г“ = „Ґ“ with strength=primary, and knowing that in Russian (but not Kyrgyz) „Е“ = „Ё“ with strength=primary, the three SELECT statements here will return results in three different orders: `CREATE TABLE things (remark STRING PRIMARY KEY); INSERT INTO things VALUES ('Е2'), ('Ё1'); INSERT INTO things VALUES ('Г2'), ('Ґ1'); SELECT remark FROM things ORDER BY remark COLLATE "unicode"; SELECT remark FROM things ORDER BY remark COLLATE "unicode_uk_s1"; SELECT remark FROM things ORDER BY remark COLLATE "unicode_ky_s1";`

#### Параметры функций по умолчанию

Starting in Tarantool 2.10, if a parameter for an *aggregate function* or a *built-in scalar SQL function* is one of the `extra-parameters` that can appear in *box.execute(...[,extra-parameters])* requests, default data type is calculated thus: * When there is only one possible data type, it is default. Example: `box.execute([[SELECT TYPEOF(LOWER(?));]],{x})` is „string“. * When possible data types are INTEGER or DOUBLE or DECIMAL, DECIMAL is default. Example: `box.execute([[SELECT TYPEOF(AVG(?));]],{x})` is „decimal“. * When possible data types are STRING or VARBINARY, STRING is default. Example: `box.execute([[SELECT TYPEOF(LENGTH(?));]],{x})` is „string“. * When possible data types are any other scalar data type, SCALAR is default. Example: `box.execute([[SELECT TYPEOF(GREATEST(?,5));]],{x})` is „scalar“. * When possible data type is a non-scalar data type, such as ARRAY, result is undefined. * Otherwise, there is no default. Example: `box.execute([[SELECT TYPEOF(LIKELY(?));]],{x})` is the name of one of the primitive data types.

### 9.4.4 SQL PLUS LUA – Adding Tarantool/NoSQL to Tarantool/SQL

The Adding Tarantool/NoSQL To Tarantool/SQL Guide contains descriptions of NoSQL database objects that can be accessed from SQL, of SQL database objects that can be accessed from NoSQL, of the way to call SQL from Lua, and of the way to call Lua from SQL.

| Heading | Введение |
|---|---|
| *Lua requests* | Some Lua requests that are especially useful for SQL, such as requests to grant privileges |
| *System tables* | Looking at Lua sysview spaces such as _space |
| *Calling Lua routines from SQL* | Tarantool's implementation of SQL stored procedures |
| *Executing Lua chunks* | The LUA(...) function |
| *Example sessions* | Million-row insert, etc. |
| *Lua functions to make views of metadata* | Making equivalents to standard-SQL information_schema tables |

#### Lua Requests

A great deal of functionality is not specifically part of Tarantool's SQL feature, but is part of the Tarantool Lua application server and DBMS. Here are some examples so it is clear where to look in other sections of the Tarantool manual.

NoSQL *«spaces»* can be accessed as SQL `"tables"`, and vice versa. For example, suppose a table has been created with `CREATE TABLE things (id INTEGER PRIMARY KEY, remark SCALAR);`

This is viewable from Tarantool's NoSQL feature as a memtx space named THINGS with a primary-key *TREE index* ...

```
tarantool> box.space.THINGS
---
- engine: memtx
  before_replace: 'function: 0x40bb4608'
  on_replace: 'function: 0x40bb45e0'
  ck_constraint: []
  field_count: 2
  temporary: false
  index:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  0: &0
    unique: true
    parts:
   - type: integer
      is_nullable: false
      fieldno: 1
    id: 0
    space_id: 520
    type: TREE
    name: pk_unnamed_THINGS_1
  pk_unnamed_THINGS_1: *0
is_local: false
enabled: true
name: THINGS
id: 520
```

The NoSQL *basic data operation requests* select, insert, replace, upsert, update, delete will all work. Particularly interesting are the requests that come only via NoSQL.

To create an index on things (remark) with a non-default *option* for example a special id, say: `box.space.THINGS:create_index('idx_100_things_2', {id=100, parts={2, 'scalar'}})`

(If the SQL data type name is SCALAR, then the NoSQL type is „scalar“, as described earlier. See the chart in section *Operands*.)

To *grant* database-access privileges to user „guest“, say `box.schema.user.grant('guest', 'execute', 'universe')` To grant SELECT privileges on table things to user „guest“, say `box.schema.user.grant('guest', 'read', 'space', 'THINGS')` To grant UPDATE privileges on table things to user „guest“, say: `box.schema.user.grant('guest', 'read,write', 'space', 'THINGS')` To grant DELETE or INSERT privileges on table things if no reading is involved, say: `box.schema.user.grant('guest', 'write', 'space', 'THINGS')` To grant DELETE or INSERT privileges on table things if reading is involved, say: `box.schema.user.grant('guest', 'read,write', 'space', 'THINGS')` To grant CREATE TABLE privilege to user „guest“, say `box.schema.user.grant('guest', 'read,write', 'space', '_schema')` `box.schema.user.grant('guest', 'read,write', 'space', '_space')` `box.schema.user.grant('guest', 'read,write', 'space', '_index')` `box.schema.user.grant('guest', 'create', 'space')` To grant CREATE TRIGGER privilege to user „guest“, say `box.schema.user.grant('guest', 'read', 'space', '_space')` `box.schema.user.grant('guest', 'read,write', 'space', '_trigger')` To grant CREATE INDEX privilege to user „guest“, say `box.schema.user.grant('guest', 'read,write', 'space', '_index')` `box.schema.user.grant('guest', 'create', 'space')` To grant CREATE TABLE ... INTEGER PRIMARY KEY AUTOINCREMENT to user „guest“, say `box.schema.user.grant('guest', 'read,write', 'space', '_schema')` `box.schema.user.grant('guest', 'read,write', 'space', '_space')` `box.schema.user.grant('guest', 'read,write', 'space', '_index')` `box.schema.user.grant('guest', 'create', 'space')` `box.schema.user.grant('guest', 'read,write', 'space', '_space_sequence')` `box.schema.user.grant('guest', 'read,write', 'space', '_sequence')` `box.schema.user.grant('guest', 'create', 'sequence')`

To write a stored procedure that inserts 5 rows in things, say `function f() for i = 3, 7 do box.space.THINGS:insert{i, i} end end` For client-side API functions, see section *«Connectors»*.

To make spaces with field names that SQL can understand, use *space_object:format()*. (Exception: in Tarantool/NoSQL it is legal for tuples to have more fields than are described by a format clause, but in Tarantool/SQL such fields will be ignored.)

To handle replication and sharding of SQL data, see section *Sharding*.

To enhance performance of SQL statements by preparing them in advance, see section *box.prepare()*.

To call SQL from Lua, see section *box.execute([[...]])*.

Limitations: (Issue#2368) * after `box.schema.user.grant('guest','read,write,execute', 'universe')`, user `'guest'` can create tables. But this is a powerful set of privileges.

Limitations: (Issue#4659, Issue#4757, Issue#4758) SELECT with * or ORDER BY or GROUP BY from spaces that have map fields or array fields may cause errors. Any access to spaces that have hash indexes may cause severe errors in Tarantool version 2.3 or earlier.

### System Tables

There is a way to get some information about the database objects, for example the names of all the tables and their indexes, using *SELECT statements*. This is done by looking at special read-only tables which Tarantool updates automatically whenever objects are created or dropped. See the *submodule box.space* overview section. Names of system tables are in lower case so always enclose them in `"quotes"`.

For example, the *_space* system table has these fields which are seen in SQL as columns: id = numeric identifier owner = for example, 1 if the object was made by the `'admin'` user name = the name that was used with *CREATE TABLE* engine = usually `'memtx'` (the `'vinyl'` engine can be used but is not default) field_count = sometimes 0, but usually a count of the table's columns flags = usually empty format = what a Lua format() function or an SQL CREATE statement produced Example selection: `SELECT "id", "name" FROM "_space";`

See also: *Lua functions to make views of metadata*.

### Calling Lua routines from SQL

SQL statements can invoke functions that are written in Lua. This is Tarantool's equivalent for the «stored procedure» feature found in other SQL DBMSs. Tarantool server-side stored procedures are written in Lua rather than SQL/PSM dialect.

Functions can be invoked anywhere that the SQL syntax allows a literal or a column name for reading. Function parameters can include any number of SQL values. If a SELECT statement's result set has a million rows, and the *select list* invokes a non-deterministic function, then the function is called a million times.

To create a Lua function that you can call from SQL, use *box.schema.func.create(func-name, {options-with-body})* with these additional options:

`exports = {'LUA', 'SQL'}` – This indicates what languages can call the function. The default is `'LUA'`. Specify both: `'LUA', 'SQL'`.

`param_list = {list}` – This is the list of parameters. Specify the Lua type names for each parameter of the function. Remember that a Lua type name is *the same as* an SQL data type name, in lower case. The Lua type should not be an array.

Also it is good to specify `{deterministic = true}` if possible, because that may allow Tarantool to generate more efficient SQL byte code.

For a useful example, here is a general function for decoding a single Lua `'map'` field:

```
box.schema.func.create('_DECODE',
   {language = 'LUA',
    returns = 'string',
    body = [[function (field, key)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            -- If Tarantool version < 2.10.1, replace next line with
            -- return require('msgpack').decode(field)[key]
            return field[key]
            end]],
    is_sandboxed = false,
    -- If Tarantool version < 2.10.1, replace next line with
    -- param_list = {'string', 'string'},
    param_list = {'map', 'string'},
    exports = {'LUA', 'SQL'},
    is_deterministic = true})
```

See it work with, say, the _trigger space. That space has a `'map'` field named opts which has a key named sql. By selecting from the space and passing the field and the key name to _DECODE, you can get a list of all the trigger bodies.

```
box.execute([[SELECT _decode("opts", 'sql') FROM "_trigger";]])
```

Remember that SQL converts *regular identifiers* to upper case, so this example works with a function named _DECODE. If the function had been named _decode, then the SELECT statement would have to be: `box.execute([[SELECT "_decode"("opts", 'sql') FROM "_trigger";]])`

Here is another example, which illustrates the way that Tarantool creates a view which includes the table_name and table_type columns in the same way that the standard-SQL information_schema.tables view contains them. The difficulty is that, in order to discover whether table_type should be `'BASE TABLE'` or should be `'VIEW'`, it is necessary to know the value of the `"flags"` field in the Tarantool/NoSQL *«_space»* or `"_vspace"` space. The `"flags"` field type is `"map"`, which SQL does not understand well. If there were no Lua functions, it would be necessary to treat the field as a VARBINARY and look for `POSITION(X'A476696577C3',"flags") > 0` (A4 is a MsgPack signal that a 4-byte string follows, 76696577 is UTF8 encoding for „view“, C3 is a MsgPack code meaning true). In any case, starting with Tarantool version 2.10, POSITION() does not work on VARBINARY operands. But there is a more sophisticated way, namely, creating a function that returns true if `"flags".view` is true. So for this case the way to make the function looks like this:

```
box.schema.func.create('TABLES_IS_VIEW',
     {language = 'LUA',
      returns = 'boolean',
      body = [[function (flags)
          local view
          -- If Tarantool version < 2.10.1, replace next line with
          -- view = require('msgpack').decode(flags).view
          view = flags.view
          if view == nil then return false end
          return view
          end]],
     is_sandboxed = false,
     -- If Tarantool version < 2.10.1, replace next line with
     -- param_list = {'string'},
     param_list = {'map'},
     exports = {'LUA', 'SQL'},
     is_deterministic = true})
```

And this creates the view:

```
box.execute([[
CREATE VIEW vtables AS SELECT
"name" AS table_name,
CASE WHEN tables_is_view("flags") == TRUE THEN 'VIEW'
     ELSE 'BASE TABLE' END AS table_type,
"id" AS id,
"engine" AS engine,
(SELECT "name" FROM "_vuser" x
 WHERE x."id" = y."owner") AS owner,
"field_count" AS field_count
FROM "_vspace" y;
]])
```

Remember that these Lua functions are persistent, so if the server has to be restarted then they do not have to be re-declared.

### Executing Lua chunks

To execute Lua code without creating a function, use: `LUA(Lua-code-string)` where Lua-code-string is any amount of Lua code. The string should begin with `'return '`.

For example this will show the number of seconds since the epoch: `box.execute([[SELECT lua('return os.time()');]])` For example this will show a database configuration member: `box.execute([[SELECT lua('return box.cfg.memtx_memory');]])` For example this will return FALSE because Lua nil and box.NULL are the same as SQL NULL: `box.execute([[SELECT lua('return box.NULL') IS NOT NULL;]])`

Warning: the SQL statement must not invoke a Lua function, or execute a Lua chunk, that accesses a space that underlies any SQL table that the SQL statement accesses. For example, if function `f()` contains a request `"box.space.TEST:insert{0}"`, then the SQL statement `"SELECT f() FROM test;"` will try to access the same space in two ways. The results of such conflict may include a hang or an infinite loop.

### Example Sessions

#### Example Session – Create, Insert, Select

Assume that the task is to create two tables, put some rows in each table, create a *view* that is based on a join of the tables, then select from the view all rows where the second column values are not null, ordered by the first column.

That is, the way to populate the table is `CREATE TABLE t1 (c1 INTEGER PRIMARY KEY, c2 STRING); CREATE TABLE t2 (c1 INTEGER PRIMARY KEY, x2 STRING);  INSERT INTO t1 VALUES (1, 'A'), (2, 'B'), (3, 'C'); INSERT INTO t1 VALUES (4, 'D'), (5, 'E'), (6, 'F'); INSERT INTO t2 VALUES (1, 'C'), (4, 'A'), (6, NULL); CREATE VIEW v AS SELECT * FROM t1 NATURAL JOIN t2; SELECT * FROM v WHERE c2 IS NOT NULL ORDER BY c1;`

So the session looks like this: `box.cfg{} box.execute([[CREATE TABLE t1 (c1 INTEGER PRIMARY KEY, c2 STRING);]]) box.execute([[CREATE TABLE t2 (c1 INTEGER PRIMARY KEY, x2 STRING);]]) box.execute([[INSERT INTO t1 VALUES (1, 'A'), (2, 'B'), (3, 'C');]]) box.execute([[INSERT INTO t1 VALUES (4, 'D'), (5, 'E'), (6, 'F');]])  box.execute([[INSERT INTO t2 VALUES (1, 'C'), (4, 'A'), (6, NULL);]]) box.execute([[CREATE VIEW v AS SELECT * FROM t1 NATURAL JOIN t2;]]) box.execute([[SELECT * FROM v WHERE c2 IS NOT NULL ORDER BY c1;]])`

If one executes the above requests with Tarantool as a client, provided the database objects do not already exist, the execution will be successful and the final display will be

```
tarantool> box.execute([[SELECT * FROM v WHERE c2 IS NOT NULL ORDER BY c1;]])
---
- - [1, 'A', 'C']
- [4, 'D', 'A']
- [6, 'F', null]
```

### Example Session – Get a List of Columns

Here is a function which will create a table that contains a list of all the columns and their Lua types, for all tables. It is not a necessary function because one can create a *_COLUMNS view* instead. It merely shows, with simpler Lua code, how to make a base table instead of a view.

```
function create_information_schema_columns()
  box.execute([[DROP TABLE IF EXISTS information_schema_columns;]])
  box.execute([[CREATE TABLE information_schema_columns (
                    table_name STRING,
                    column_name STRING,
                    ordinal_position INTEGER,
                    data_type STRING,
                    PRIMARY KEY (table_name, column_name));]]);
  local space = box.space._vspace:select()
  local sqlstring = ''
  for i = 1, #space do
      for j = 1, #space[i][7] do
          sqlstring = "INSERT INTO information_schema_columns VALUES ("
                  .. "'" .. space[i][3] .. "'"
                  .. ","
                  .. "'" .. space[i][7][j].name .. "'"
                  .. ","
                  .. j
                  .. ","
                  .. "'" .. space[i][7][j].type .. "'"
                  .. ");"
          box.execute(sqlstring)
      end
  end
  return
end
```

If you now execute the function by saying `create_information_schema_columns()` you will see that there is a table named information_schema_columns containing table_name and column_name and ordinal_position and data_type for everything that was accessible.

### Example Session – Million-Row Insert

Here is a variation of the Lua tutorial *«Insert one million tuples with a Lua stored procedure»*. The differences are: the creation is done with an SQL *CREATE TABLE statement*, and the inserting is done with an SQL *INSERT statement*. Otherwise, it is the same. It is the same because Lua and SQL are compatible, just as Lua and NoSQL are compatible.

```
box.execute([[CREATE TABLE tester (s1 INTEGER PRIMARY KEY, s2 STRING);]])

function string_function()
  local random_number
  local random_string
  random_string = ""
  for x = 1,10,1 do
    random_number = math.random(65, 90)
    random_string = random_string .. string.char(random_number)
  end
  return random_string
end

function main_function()
    local string_value, t, sql_statement
    for i = 1,1000000, 1 do
    string_value = string_function()
    sql_statement = "INSERT INTO tester VALUES (" .. i .. ",'" .. string_value .. "');"
    box.execute(sql_statement)
    end
end
start_time = os.clock()
main_function()
end_time = os.clock()
'insert done in ' .. end_time - start_time .. ' seconds'
```

Limitations: The function takes more time than the original (Tarantool/NoSQL).

### Lua functions to make views of metadata

Tarantool does not include all the standard-SQL information_schema views, which are for looking at metadata, that is, «data about the data». But here is the Lua code and SQL code for creating equivalents: *_TABLES* nearly equivalent to INFORMATION_SCHEMA.TABLES *_COLUMNS* nearly equivalent to INFORMATION_SCHEMA.COLUMNS *_VIEWS* nearly equivalent to INFORMATION_SCHEMA.VIEWS *_TRIGGERS* nearly equivalent to INFORMATION_SCHEMA.TRIGGERS *_REFERENTIAL_CONSTRAINTS* nearly equivalent to INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS *_CHECK_CONSTRAINTS* nearly equivalent to INFORMATION_SCHEMA.CHECK_CONSTRAINTS *_TABLE_CONSTRAINTS* nearly equivalent to INFORMATION_SCHEMA.TABLE_CONSTRAINTS. For each view there will be an example of a SELECT from the view, and the code. Users who want metadata can simply copy the code. Use this code only with Tarantool version 2.3.0 or later. With an earlier Tarantool version, a *PRAGMA statement* may be useful.

### _TABLES view

Пример:

```
tarantool>SELECT * FROM _tables WHERE id > 340 LIMIT 5;
OK 5 rows selected (0.0 seconds)
+---------------+-------------+---------------+-----------+-----+--------+-------+----
↪---------+
| TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME     | TABLE_TYPE | ID  | ENGINE | OWNER |␣
↪FIELD_COUNT |
+---------------+-------------+---------------+-----------+-----+--------+-------+----
↪---------+
| NULL          | NULL         | _fk_constraint | BASE TABLE | 356 | memtx  | admin |  ␣
↪    0    |
| NULL          | NULL         | _ck_constraint | BASE TABLE | 364 | memtx  | admin |  ␣
↪    0    |
| NULL          | NULL         | _func_index    | BASE TABLE | 372 | memtx  | admin |  ␣
↪    0    |
| NULL          | NULL         | _COLUMNS       | VIEW       | 513 | memtx  | admin |  ␣
↪    8    |
| NULL          | NULL         | _VIEWS         | VIEW       | 514 | memtx  | admin |  ␣
↪    7    |
+---------------+-------------+---------------+-----------+-----+--------+-------+----
↪---------+
```

Definition of the function and the CREATE VIEW statement:

```
box.schema.func.drop('_TABLES_IS_VIEW',{if_exists = true})
box.schema.func.create('_TABLES_IS_VIEW',
     {language = 'LUA',
      returns = 'boolean',
      body = [[function (flags)
          local view
          -- If Tarantool version < 2.10.1, replace next line with
          -- view = require('msgpack').decode(flags).view
          view = flags.view
          if view == nil then return false end
          return view
          end]],
     is_sandboxed = false,
     -- If Tarantool version < 2.10.1, replace next line with
     -- param_list = {'string'},
     param_list = {'map'},
     exports = {'LUA', 'SQL'},
     is_deterministic = true})
box.schema.role.grant('public', 'execute', 'function', '_TABLES_IS_VIEW')
pcall(function ()
    box.schema.role.revoke('public', 'read', 'space', '_TABLES', {if_exists = true})
    end)
box.execute([[DROP VIEW IF EXISTS _tables;]])
box.execute([[
CREATE VIEW _tables AS SELECT
    CAST(NULL AS STRING) AS table_catalog,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    CAST(NULL AS STRING) AS table_schema,
    "name" AS table_name,
    CASE
        WHEN _tables_is_view("flags") = TRUE THEN 'VIEW'
        ELSE 'BASE TABLE' END
        AS table_type,
    "id" AS id,
    "engine" AS engine,
    (SELECT "name" FROM "_vuser" x WHERE x."id" = y."owner") AS owner,
    "field_count" AS field_count
FROM "_vspace" y;
]])
box.schema.role.grant('public', 'read', 'space', '_TABLES')
```

### _COLUMNS view

Этот пример также показывает, как с помощью *рекурсивных представлений* создавать временные таблицы с несколькими строками для каждого кортежа в исходном спейсе `"_vspace"`. Для этого потребуется глобальная переменная `_G.box.FORMATS` в качестве временной статической переменной.

Warning: Use this code only with Tarantool version 2.3.2 or later. Use with earlier versions will cause an assertion. See Issue#4504.

Пример:

```
tarantool>SELECT * FROM _columns WHERE ordinal_position = 9;
OK 6 rows selected (0.0 seconds)
+--------------+-------------+--------------------------+--------------+-----------------
↪-+-------------+-----------+-----+
| CATALOG_NAME | SCHEMA_NAME | TABLE_NAME               | COLUMN_NAME  | ORDINAL_
↪POSITION | IS_NULLABLE | DATA_TYPE | ID  |
+--------------+-------------+--------------------------+--------------+-----------------
↪-+-------------+-----------+-----+
| NULL         | NULL        | _sequence                | cycle        |               ␣
↪9 | YES          | boolean   | 284 |
| NULL         | NULL        | _vsequence               | cycle        |               ␣
↪9 | YES          | boolean   | 286 |
| NULL         | NULL        | _func                    | returns      |               ␣
↪9   YES            string    | 296 |
| NULL         | NULL        | _fk_constraint           | parent_cols  |               ␣
↪9 | YES          | array     | 356 |
| NULL         | NULL        | _REFERENTIAL_CONSTRAINTS | MATCH_OPTION |               ␣
↪9 | YES          | string    | 518 |
+--------------+-------------+--------------------------+--------------+-----------------
↪-+-------------+-----------+-----+
```

Definition of the function and the CREATE VIEW statement:

```
box.schema.func.drop('_COLUMNS_FORMATS', {if_exists = true})
box.schema.func.create('_COLUMNS_FORMATS',
    {language = 'LUA',
     returns = 'scalar',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    body = [[
    function (row_number_, ordinal_position)
        if row_number_ == 0 then
            _G.box.FORMATS = {}
            local vspace = box.space._vspace:select()
            for i = 1, #vspace do
                local format = vspace[i]["format"]
                for j = 1, #format do
                    local is_nullable = 'YES'
                    if format[j].is_nullable == false then
                        is_nullable = 'NO'
                    end
                    table.insert(_G.box.FORMATS,
                                 {vspace[i].name, format[j].name, j,
                                  is_nullable, format[j].type, vspace[i].id})
                end
            end
            return ''
        end
        if row_number_ > #_G.box.FORMATS then
            _G.box.FORMATS = {}
            return ''
        end
        return _G.box.FORMATS[row_number_][ordinal_position]
    end
    ]],
    param_list = {'integer', 'integer'},
    exports = {'LUA', 'SQL'},
    is_sandboxed = false,
    setuid = false,
    is_deterministic = false})
box.schema.role.grant('public', 'execute', 'function', '_COLUMNS_FORMATS')

pcall(function ()
    box.schema.role.revoke('public', 'read', 'space', '_COLUMNS', {if_exists = true})
    end)
box.execute([[DROP VIEW IF EXISTS _columns;]])
box.execute([[
CREATE VIEW _columns AS
WITH RECURSIVE r_columns AS
(
SELECT 0 AS row_number_,
      '' AS table_name,
      '' AS column_name,
      0 AS ordinal_position,
      '' AS is_nullable,
      '' AS data_type,
      0 AS id
UNION ALL
SELECT row_number_ + 1 AS row_number_,
       _columns_formats(row_number_, 1) AS table_name,
       _columns_formats(row_number_, 2) AS column_name,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
       _columns_formats(row_number_, 3) AS ordinal_position,
       _columns_formats(row_number_, 4) AS is_nullable,
       _columns_formats(row_number_, 5) AS data_type,
       _columns_formats(row_number_, 6) AS id
    FROM r_columns
    WHERE row_number_ == 0 OR row_number_ <= lua('return #_G.box.FORMATS + 1')
)
SELECT CAST(NULL AS STRING) AS catalog_name,
       CAST(NULL AS STRING) AS schema_name,
       table_name,
       column_name,
       ordinal_position,
       is_nullable,
       data_type,
       id
    FROM r_columns
    WHERE data_type <> '';
]])
box.schema.role.grant('public', 'read', 'space', '_COLUMNS')
```

### _VIEWS view

Пример:

```
tarantool>SELECT table_name, substr(view_definition,1,20), id, owner, field_count FROM _
↪views LIMIT 5;
OK 5 rows selected (0.0 seconds)
+--------------------------+------------------------------+-----+-------+-------------+
| TABLE_NAME               | SUBSTR(VIEW_DEFINITION,1,20) | ID  | OWNER | FIELD_COUNT |
+--------------------------+------------------------------+-----+-------+-------------+
| _COLUMNS                 | CREATE VIEW _columns         | 513 | admin |           8 |
| _TRIGGERS                | CREATE VIEW _trigger         | 515 | admin |           4 |
| _CHECK_CONSTRAINTS       | CREATE VIEW _check_c         | 517 | admin |           8 |
| _REFERENTIAL_CONSTRAINTS | CREATE VIEW _referen         | 518 | admin |          12 |
| _TABLE_CONSTRAINTS       | CREATE VIEW _table_c         | 519 | admin |          11 |
+--------------------------+------------------------------+-----+-------+-------------+
```

Definition of the function and the CREATE VIEW statement:

```
box.schema.func.drop('_VIEWS_DEFINITION',{if_exists = true})
box.schema.func.create('_VIEWS_DEFINITION',
    {language = 'LUA',
     returns = 'string',
     body = [[function (flags)
              -- If Tarantool version < 2.10.1, replace next line with
              -- return require('msgpack').decode(flags).sql
              return flags.sql
              end]],
     -- If Tarantool version < 2.10.1, replace next line with
     -- param_list = {'string'},
     param_list = {'map'},
```

(continues on next page)

(продолжение с предыдущей страницы)

```
     exports = {'LUA', 'SQL'},
     is_sandboxed = false,
     setuid = false,
     is_deterministic = false})
box.schema.role.grant('public', 'execute', 'function', '_VIEWS_DEFINITION')
pcall(function ()
    box.schema.role.revoke('public', 'read', 'space', '_VIEWS', {if_exists = true})
    end)
box.execute([[DROP VIEW IF EXISTS _views;]])
box.execute([[
CREATE VIEW _views AS SELECT
    CAST(NULL AS STRING) AS table_catalog,
    CAST(NULL AS STRING) AS table_schema,
    "name" AS table_name,
    CAST(_views_definition("flags") AS STRING) AS VIEW_DEFINITION,
    "id" AS id,
    (SELECT "name" FROM "_vuser" x WHERE x."id" = y."owner") AS owner,
    "field_count" AS field_count
    FROM "_vspace" y
    WHERE _tables_is_view("flags") = TRUE;
]])
box.schema.role.grant('public', 'read', 'space', '_VIEWS')
```

_TABLES_IS_VIEW() was described earlier, see *_TABLES view*.

### _TRIGGERS view

Пример:

```
tarantool>SELECT trigger_name, opts_sql FROM _triggers;
OK 2 rows selected (0.0 seconds)
+--------------+-------------------------------------------------------------------------------
↪------------------------+
| TRIGGER_NAME | OPTS_SQL                                                                      ␣
↪                        |
+--------------+-------------------------------------------------------------------------------
↪------------------------+
| THINGS1_AD   | CREATE TRIGGER things1_ad AFTER DELETE ON things1 FOR EACH ROW BEGIN␣
↪DELETE FROM things2; END;  |
| THINGS1_BI   | CREATE TRIGGER things1_bi BEFORE INSERT ON things1 FOR EACH ROW BEGIN␣
↪DELETE FROM things2; END; |
+--------------+-------------------------------------------------------------------------------
↪------------------------+
```

Definition of the function and the CREATE VIEW statement:

```
box.schema.func.drop('_TRIGGERS_OPTS_SQL',{if_exists = true})
box.schema.func.create('_TRIGGERS_OPTS_SQL',
    {language = 'LUA',
     returns = 'string',
     body = [[function (opts)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            -- If Tarantool version < 2.10.1, replace next line with
            -- return require('msgpack').decode(opts).sql
            return opts.sql
            end]],
    -- If Tarantool version < 2.10.1, replace next line with
    -- param_list = {'string'},
    param_list = {'map'},
    exports = {'LUA', 'SQL'},
    is_sandboxed = false,
    setuid = false,
    is_deterministic = false})
box.schema.role.grant('public', 'execute', 'function', '_TRIGGERS_OPTS_SQL')
pcall(function ()
    box.schema.role.revoke('public', 'read', 'space', '_TRIGGERS', {if_exists = true})
    end)
box.execute([[DROP VIEW IF EXISTS _triggers;]])
box.execute([[
CREATE VIEW _triggers AS SELECT
    CAST(NULL AS STRING) AS trigger_catalog,
    CAST(NULL AS STRING) AS trigger_schema,
    "name" AS trigger_name,
    CAST(_triggers_opts_sql("opts") AS STRING) AS opts_sql,
    "space_id" AS space_id
    FROM "_trigger";
]])
box.schema.role.grant('public', 'read', 'space', '_TRIGGERS')
```

Users who select from this view will need „read“ privilege on the _trigger space.

### _REFERENTIAL_CONSTRAINTS view

Пример:

```
tarantool>SELECT constraint_name, update_rule, delete_rule, match_option,
> referencing, referenced
> FROM _referential_constraints;
OK 2 rows selected (0.0 seconds)
+----------------------+-------------+-------------+--------------+-------------+--------
↪----+
| CONSTRAINT_NAME      | UPDATE_RULE | DELETE_RULE | MATCH_OPTION | REFERENCING |␣
↪REFERENCED |
+----------------------+-------------+-------------+--------------+-------------+--------
↪----+
| fk_unnamed_THINGS2_1 | no_action   | no_action   | simple       | THINGS2     |␣
↪THINGS1    |
| fk_unnamed_THINGS3_1 | no_action   | no_action   | simple       | THINGS3     |␣
↪THINGS1    |
+----------------------+-------------+-------------+--------------+-------------+--------
↪----+
```

Definition of the CREATE VIEW statement:

```
pcall(function ()
    box.schema.role.revoke('public', 'read', 'space', '_REFERENTIAL_CONSTRAINTS', {if_
↪exists = true})
    end)
box.execute([[DROP VIEW IF EXISTS _referential_constraints;]])
box.execute([[
CREATE VIEW _referential_constraints AS SELECT
    CAST(NULL AS STRING) AS constraint_catalog,
    CAST(NULL AS STRING) AS constraint_schema,
    "name" AS constraint_name,
    CAST(NULL AS STRING) AS unique_constraint_catalog,
    CAST(NULL AS STRING) AS unique_constraint_schema,
    '' AS unique_constraint_name,
    "on_update" AS update_rule,
    "on_delete" AS delete_rule,
    "match" AS match_option,
    (SELECT "name" FROM "_vspace" x WHERE x."id" = y."child_id") AS referencing,
    (SELECT "name" FROM "_vspace" x WHERE x."id" = y."parent_id") AS referenced,
    "is_deferred" AS is_deferred,
    "child_id" AS child_id,
    "parent_id" AS parent_id
    FROM "_fk_constraint" y;
]])
box.schema.role.grant('public', 'read', 'space', '_REFERENTIAL_CONSTRAINTS')
```

In this example child_cols or parent_cols are not taken from the _fk_constraint space because in standard SQL those are in a separate table.

Users who select from this view will need „read“ privilege on the _fk_constraint space.

### _CHECK_CONSTRAINTS view

Пример:

```
tarantool>SELECT constraint_name, check_clause, space_name, language
> FROM _check_constraints;
OK 3 rows selected (0.0 seconds)
+-----------------------+-------------------------+------------+----------+
| CONSTRAINT_NAME       | CHECK_CLAUSE            | SPACE_NAME | LANGUAGE |
+-----------------------+-------------------------+------------+----------+
| ck_unnamed_Employees_1 | first_name LIKE 'Влад%' | Employees  | SQL      |
| ck_unnamed_Critics_1   | first_name LIKE 'Vlad%' | Critics    | SQL      |
| ck_unnamed_ACTORS_1    | salary > 0              | ACTORS     | SQL      |
+-----------------------+-------------------------+------------+----------+
```

Definition of the CREATE VIEW statement:

```
pcall(function ()
    box.schema.role.revoke('public', 'read', 'space', '_CHECK_CONSTRAINTS', {if_exists =␣
↪true})
    end)
box.execute([[DROP VIEW IF EXISTS _check_constraints;]])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
box.execute([[
CREATE VIEW _check_constraints AS SELECT
    CAST(NULL AS STRING) AS constraint_catalog,
    CAST(NULL AS STRING) AS constraint_schema,
    "name" AS constraint_name,
    "code" AS check_clause,
    (SELECT "name" FROM "_vspace" x WHERE x."id" = y."space_id") AS space_name,
    "language" AS language,
    "is_deferred" AS is_deferred,
    "space_id" AS space_id
    FROM "_ck_constraint" y;
]])
box.schema.role.grant('public', 'read', 'space', '_CHECK_CONSTRAINTS')
```

Users who select from this view will need „read“ privilege on the _ck_constraint space.

### _TABLE_CONSTRAINTS view

This has only the constraints (primary-key and unique-key) that can be found by looking at the *_index* space. It is not a list of indexes, that is, it is not equivalent to INFORMATION_SCHEMA.STATISTICS. The columns of the index are not taken because in standard SQL they would be in a different table.

Пример:

```
tarantool>SELECT constraint_name, constraint_type, table_name, id, iid, index_type
> FROM _table_constraints
> LIMIT 5;
OK 5 rows selected (0.0 seconds)
+-----------------+-----------------+-------------+-----+-----+------------+
| CONSTRAINT_NAME | CONSTRAINT_TYPE | TABLE_NAME  | ID  | IID | INDEX_TYPE |
+-----------------+-----------------+-------------+-----+-----+------------+
| primary         | PRIMARY         | _schema     | 272 |   0 | tree       |
| primary         | PRIMARY         | _collation  | 276 |   0 | tree       |
| name            | UNIQUE          | _collation  | 276 |   1 | tree       |
| primary         | PRIMARY         | _vcollation | 277 |   0 | tree       |
| name            | UNIQUE          | _vcollation | 277 |   1 | tree       |
+-----------------+-----------------+-------------+-----+-----+------------+
```

Definition of the function and the CREATE VIEW statement:

```
box.schema.func.drop('_TABLE_CONSTRAINTS_OPTS_UNIQUE',{if_exists = true})
function _TABLE_CONSTRAINTS_OPTS_UNIQUE (opts) return require('msgpack').decode(opts).
↪unique end
box.schema.func.create('_TABLE_CONSTRAINTS_OPTS_UNIQUE',
    {language = 'LUA',
     returns = 'boolean',
     body = [[function (opts) return require('msgpack').decode(opts).unique end]],
     param_list = {'string'},
     exports = {'LUA', 'SQL'},
     is_sandboxed = false,
     setuid = false,
     is_deterministic = false})
```

(continues on next page)

(продолжение с предыдущей страницы)

```
box.schema.role.grant('public', 'execute', 'function', '_TABLE_CONSTRAINTS_OPTS_UNIQUE')
pcall(function ()
box.schema.role.revoke('public', 'read', 'space', '_TABLE_CONSTRAINTS', {if_exists =␣
↪true})
end)
box.execute([[DROP VIEW IF EXISTS _table_constraints;]])
box.execute([[
CREATE VIEW _table_constraints AS SELECT
CAST(NULL AS STRING) AS constraint_catalog,
CAST(NULL AS STRING) AS constraint_schema,
"name" AS constraint_name,
(SELECT "name" FROM "_vspace" x WHERE x."id" = y."id") AS table_name,
CASE WHEN "iid" = 0 THEN 'PRIMARY' ELSE 'UNIQUE' END AS constraint_type,
CAST(NULL AS STRING) AS initially_deferrable,
CAST(NULL AS STRING) AS deferred,
CAST(NULL AS STRING) AS enforced,
"id" AS id,
"iid" AS iid,
"type" AS index_type
FROM "_vindex" y
WHERE _table_constraints_opts_unique("opts") = TRUE;
]])
box.schema.role.grant('public', 'read', 'space', '_TABLE_CONSTRAINTS')
```

### 9.4.5 Возможности SQL

This section compares Tarantool's features with SQL:2016's «Feature taxonomy and definition for mandatory features».

For each feature in that list, there will be a simple example SQL statement. If Tarantool appears to handle the example, it will be marked «Okay», else it will be marked «Fail». Since this is rough and arbitrary, the hope is that tests which are unfairly marked «Okay» will probably be balanced by tests which are unfairly marked «Fail».

### E011, числовые типы данных

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E011-01 | Типы данных INTEGER и SMALLINT | `CREATE TABLE t (s1 INT PRIMARY KEY);` | *Работает*. |
| E011-02 | Типы данных REAL, DOUBLE PRECISION и FLOAT | `CREATE TABLE tr (s1 FLOAT PRIMARY KEY);` | Ошибка. Тип данных с плавающей точкой в Tarantool — *DOUBLE*. **Примечание:** Скорее всего, типы с плавающей точкой в версиях 2.1 и 2.2 не будут совместимы: в версии 2.1 для столбцов этих типов задается формат „number“, а в версии 2.2 — „float32“ и „float64“. Для смены формата требуется перенос данных, который не выполняется автоматически, поскольку в версии 2.1 нет информации, как отличать столбцы „number“, созданные через Lua, от FLOAT/DOUBLE/REAL, созданных через SQL. |
| E011-03 | Типы данных DECIMAL и NUMERIC | `CREATE TABLE td (s1 NUMERIC PRIMARY KEY);` | Fail, NUMERIC data types are not supported, although the DECIMAL data type is supported. |
| E011-04 | Арифметические операторы | `SELECT 10+1, 9-2, 8*3, 7/2 FROM t;` | *Работает*. |
| E011-05 | Числовые сравнения | `SELECT * FROM t WHERE 1 < 2;` | *Работает*. |
| E011-06 | Неявное приведение числовых типов данных | `SELECT * FROM t WHERE s1 = 1.00;` | Работает, потому что Tarantool поддерживает сравнение 1.00 с INTEGER. |

### E021, типы символьных строк

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E021-01 | Символьный тип данных (включая все варианты написания) | `CREATE TABLE t44 (s1 CHAR PRIMARY KEY);` | Ошибка, тип данных CHAR не поддерживается. Этот тип ошибки будет учитываться только один раз. |
| E021-02 | Тип данных CHARACTER VARYING (включая все варианты написания) | `CREATE TABLE t45 (s1 VARCHAR PRIMARY KEY);` | Ошибка, Tarantool поддерживает только VARCHAR(n), являющийся синонимом *STRING*. |
| E021-03 | Символьные литералы | `INSERT INTO t45 VALUES ('');` | Работает. Кроме того, избегается нежелательная практика принятия "" за символьные литералы. |
| E021-04 | Функция CHARACTER_LENGTH | `SELECT character_length(s1) FROM t;` | Работает. Tarantool воспринимает эту функцию как синоним *LENGTH()*. |
| E021-05 | OCTET_LENGTH | `SELECT octet_length(s1) FROM t;` | Ошибка. Такой функции нет. |
| E021-06 | Функция SUBSTRING | `SELECT substring(s1 FROM 1 FOR 1) FROM t;` | Ошибка, такой функции нет. Есть функция *SUBSTR(x,n,n)*. |
| E021-07 | Конкатенация символов | `SELECT 'a' \|\| 'b' FROM t;` | *Работает*. |
| E021-08 | Функции UPPER и LOWER | `SELECT upper('a'), lower('B') FROM t;` | Работает. Tarantool поддерживает как *UPPER()*, так и *LOWER()*. |
| E021-09 | Функция TRIM | `SELECT trim('a ') FROM t;` | *Работает*. |
| E021-10 | Неявное приведение типов символьных строк фиксированной и переменной длины | `SELECT * FROM tm WHERE char_column > varchar_column;` | Ошибка, тип символьной строки фиксированной длины отсутствует. |
| E021-11 | Функция POSITION | `SELECT position(x IN y) FROM z;` | Ошибка. В функции Tarantool *POSITION* используется „,“, а не „IN“. |
| E021-12 | Сравнение символов | `SELECT * FROM t WHERE s1 > 'a';` | Работает. Стоит отметить, что в сравнениях по умолчанию используется двоичная сортировка, но можно использовать и *COLLATE*. |

### E031, идентификаторы

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E031 | Идентификаторы | `CREATE TABLE rank (ceil INT PRIMARY KEY);` | Ошибка. Список *зарезервированных слов* в Tarantool отличается от стандартного. |
| E031-01 | Идентификаторы с разделителем | `CREATE TABLE "t47" (s1 INT PRIMARY KEY);` | *Работает*. Кроме того, заключение идентификаторов в двойные кавычки означает, что они не будут приводиться к верхнему или нижнему регистру. В некоторых других СУБД эта особенность отсутствует. |
| E031-02 | Идентификаторы в нижнем регистре | `CREATE TABLE t48 (s1 INT PRIMARY KEY);` | Работает. |
| E031-03 | Символ нижнего подчеркивания в конце | `CREATE TABLE t49_ (s1 INT PRIMARY KEY);` | Работает. |

### E051, базовые запросы

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E051-01 | SELECT DISTINCT | `SELECT DISTINCT s1 FROM t;` | Работает. |
| E051-02 | Предложение GROUP BY | `SELECT DISTINCT s1 FROM t GROUP BY s1;` | *Работает*. |
| E051-04 | GROUP BY может содержать столбцы вне выборки SELECT | `SELECT s1 FROM t GROUP BY lower(s1);` | Работает. |
| E051-05 | Элементы в списке выборки можно переименовывать | `SELECT s1 AS K FROM t ORDER BY K;` | Работает. |
| E051-06 | Предложение HAVING | `SELECT count(*) FROM t HAVING count(*) > 0;` | Работает. Tarantool поддерживает *HAVING*, при этом GROUP BY перед HAVING не является обязательным. |
| E051-07 | Допускается использование * в квалификаторе для списка выборки | `SELECT t.* FROM t;` | Работает. |
| E051-08 | Корреляционные имена в предложении FROM | `SELECT * FROM t AS K;` | Работает. |
| E051-09 | Переименование столбцов в предложении FROM | `SELECT * FROM t AS x(q, c);` | Ошибка. |

### E061, базовые предикаты и условия поиска

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E061-01 | Предикат сравнения | `SELECT * FROM t WHERE 0 = 0;` | Работает. |
| E061-02 | Предикат BETWEEN | `SELECT * FROM t WHERE ' ' BETWEEN '' AND ' ';` | *Работает*. |
| E061-03 | Предикат IN со списком значений | `SELECT * FROM t WHERE s1 IN ('a', upper('a'));` | Работает. |
| E061-04 | Предикат LIKE | `SELECT * FROM t WHERE s1 LIKE '_';` | *Работает*. |
| E061-05 | Предикат LIKE: предложение ESCAPE | `VALUES ('abc_' LIKE 'abcX_' ESCAPE 'X');` | Работает. |
| E061-06 | Предикат NULL | `SELECT * FROM t WHERE s1 IS NOT NULL;` | *Работает*. |
| E061-07 | Предикат количественного сравнения | `SELECT * FROM t WHERE s1 = ANY (SELECT s1 FROM t);` | Ошибка — в синтаксисе. |
| E061-08 | Предикат EXISTS | `SELECT * FROM t WHERE NOT EXISTS (SELECT * FROM t);` | *Работает*. |
| E061-09 | Подзапросы в предикате сравнения | `SELECT * FROM t WHERE s1 > (SELECT s1 FROM t);` | *Работает*. |
| E061-11 | Подзапросы в предикате IN | `SELECT * FROM t WHERE s1 IN (SELECT s1 FROM t);` | Работает. |
| E061-12 | Подзапросы в предикате количественного сравнения | `SELECT * FROM t WHERE s1 >= ALL (SELECT s1 FROM t);` | Ошибка — в синтаксисе. |
| E061-13 | Коррелирующие подзапросы | `SELECT * FROM t WHERE s1 = (SELECT s1 FROM t2 WHERE t2.s2 = t.s1);` | Работает. |
| E061-14 | Условие поиска | `SELECT * FROM t WHERE 0 <> 0 OR 'a' < 'b' AND s1 IS NULL;` | Работает. |

### E071, базовые выражения с запросами

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E071-01 | Табличный оператор UNION DISTINCT | `SELECT * FROM t UNION DISTINCT SELECT * FROM t;` | Ошибка. Однако `SELECT * FROM t UNION SELECT * FROM t;` работает. |
| E071-02 | Табличный оператор UNION ALL | `SELECT * FROM t UNION ALL SELECT * FROM t;` | *Работает*. |
| E071-03 | Табличный оператор EXCEPT DISTINCT | `SELECT * FROM t EXCEPT DISTINCT SELECT * FROM t;` | Ошибка. Однако `SELECT * FROM t EXCEPT SELECT * FROM t;` работает. |
| E071-05 | Столбцы, совмещенные с помощью табличных операторов, необязательно должны иметь идентичный тип данных | `SELECT s1 FROM t UNION SELECT 5 FROM t;` | Работает. |
| E071-06 | Табличные операторы в подзапросах | `SELECT * FROM t WHERE 'a' IN (SELECT * FROM t UNION SELECT * FROM t);` | Работает. |

### E081, базовые права

Tarantool не поддерживает права, кроме как через NoSQL.

### E091, агрегатные функции

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E091-01 | AVG | `SELECT avg(s1) FROM t7;` | Ошибка. Tarantool поддерживает *AVG*, но предупреждение об исключении NULL отсутствует. |
| E091-02 | COUNT | `SELECT count(*) FROM t7 WHERE s1 > 0;` | *Работает*. |
| E091-03 | MAX | `SELECT max(s1) FROM t7 WHERE s1 > 0;` | *Работает*. |
| E091-04 | MIN | `SELECT min(s1) FROM t7 WHERE s1 > 0;` | *Работает*. |
| E091-05 | SUM | `SELECT sum(1) FROM t7 WHERE s1 > 0;` | *Работает*. |
| E091-06 | Квантификатор ALL | `SELECT sum(ALL s1) FROM t7 WHERE s1 > 0;` | Работает. |
| E091-07 | Квантификатор DISTINCT | `SELECT sum(DISTINCT s1) FROM t7 WHERE s1 > 0;` | Работает. |

### E101, базовая обработка данных

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E101-01 | Инструкция INSERT | `INSERT INTO t (s1,s2) VALUES (1,''), (2,NULL), (3,55);` | *Работает.* |
| E101-03 | Инструкция UPDATE с поиском | `UPDATE t SET s1 = NULL WHERE s1 IN (SELECT s1 FROM t2);` | *Работает.* |
| E101-04 | Инструкция DELETE с поиском | `DELETE FROM t WHERE s1 IN (SELECT s1 FROM t);` | *Работает.* |

### E111, инструкция SELECT, возвращающая одну строку

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E111 | Инструкция SELECT, возвращающая одну строку | `SELECT count(*) FROM t;` | *Работает.* |

### E121, базовая поддержка курсоров

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E121-01 | DECLARE CURSOR | | Ошибка. Tarantool не поддерживает курсоры. |
| E121-02 | Столбцы ORDER BY необязательно должны быть в списке выборки | `SELECT s1 FROM t ORDER BY s2;` | *Работает*. |
| E121-03 | Выражения в ORDER BY | `SELECT s1 FROM t7 ORDER BY -s1;` | Работает. |
| E121-04 | Инструкция OPEN | | Ошибка. Tarantool не поддерживает курсоры. |
| E121-06 | Инструкция UPDATE с позицией | | Ошибка. Tarantool не поддерживает курсоры. |
| E121-07 | Инструкция DELETE с позицией | | Ошибка. Tarantool не поддерживает курсоры. |
| E121-08 | Инструкция CLOSE | | Ошибка. Tarantool не поддерживает курсоры. |
| E121-10 | Инструкция FETCH с неявным NEXT | | Ошибка. Tarantool не поддерживает курсоры. |
| E121-17 | Курсоры WITH HOLD | | Ошибка. Tarantool не поддерживает курсоры. |

### E131, поддержка значения Null

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E131 | Поддержка значения NULL (NULL вместо значений) | `SELECT s1 FROM t7 WHERE s1 IS NULL;` | Работает. |

### E141, базовые ограничения целостности

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E141-01 | Ограничения NOT NULL | `CREATE TABLE t8 (s1 INT PRIMARY KEY, s2 INT NOT NULL);` | *Работает*. |
| E141-02 | Ограничения UNIQUE для столбцов NOT NULL | `CREATE TABLE t9 (s1 INT PRIMARY KEY , s2 INT NOT NULL UNIQUE);` | *Работает*. |
| E141-03 | Ограничения PRIMARY KEY | `CREATE TABLE t10 (s1 INT PRIMARY KEY);` | Работает, хотя в Tarantool необязательно наличие первичного ключа. |
| E141-04 | Базовое ограничение FOREIGN KEY без действия (NO ACTION) по умолчанию для операций удаления и изменения со ссылками | `CREATE TABLE t11 (s0 INT PRIMARY KEY, s1 INT REFERENCES t10);` | *Работает*. |
| E141-06 | Ограничения CHECK | `CREATE TABLE t12 (s1 INT PRIMARY KEY, s2 INT, CHECK (s1 = s2));` | Работает. |
| E141-07 | Значения столбцов по умолчанию | `CREATE TABLE t13 (s1 INT PRIMARY KEY, s2 INT DEFAULT -1);` | Работает. |
| E141-08 | NOT NULL подразумевается для первичного ключа | `CREATE TABLE t14 (s1 INT PRIMARY KEY);` | Работает. NULL вставить не удастся, хотя столбец не объявлен явным образом как NOT NULL. |
| E141-10 | Имена для FOREIGN KEY можно указывать в любом порядке | `CREATE TABLE t15 (s1 INT, s2 INT, PRIMARY KEY (s1,s2)); CREATE TABLE t16 (s1 INT PRIMARY KEY, s2 INT, FOREIGN KEY (s2,s1) REFERENCES t15 (s1,s2));` | Работает. |

### E151, поддержка транзакций

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E151-01 | Инструкция COMMIT | `COMMIT;` | Ошибка. Tarantool поддерживает *COMMIT*, но перед этим требуется использовать *START TRANSACTION*. |
| E151-02 | Инструкция ROLLBACK | `ROLLBACK;` | *Работает*. |

### E152, базовая инструкция SET TRANSACTION

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E152-01 | Инструкция SET TRANSACTION: предложение ISOLATION SERIALIZABLE | `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;` | Ошибка — в синтаксисе. |
| E152-02 | Инструкция SET TRANSACTION: предложения READ ONLY и READ WRITE | `SET TRANSACTION READ ONLY;` | Ошибка — в синтаксисе. |

### E*, прочее

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| E153 | Запросы, изменяющие данные, с подзапросами | `UPDATE "view_containing_subquery" SET column1=0;` | Ошибка. |
| E161 | Комментарии SQL с двумя знаками минуса в начале | `--comment;` | *Работает*. |
| E171 | Поддержка SQLSTATE | `DROP TABLE no_such_table;` | Ошибка. Tarantool возвращает сообщение об ошибке, но не строку SQLSTATE. |
| E182 | Привязка базового языка | | Работает. Любые коннекторы Tarantool должны иметь возможность вызвать *box.execute()*. |

### F021, базовая информационная схема

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F021 | Базовая информационная схема | `SELECT * FROM information_schema.tables;` | Ошибка. Метаданные Tarantool не содержатся в схеме с таким именем (ошибка не учитывается в итоговом подсчете). |

### F031, базовые манипуляции со схемой

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F031-01 | Инструкция CREATE TABLE создает хранимые таблицы БД | `CREATE TABLE t20 (t20_1 INT NOT NULL);` | Ошибка. PRIMARY KEY всегда должен быть указан (эта ошибка будет учитываться только один раз). |
| F031-02 | Инструкция CREATE VIEW | `CREATE VIEW t21 AS SELECT * FROM t20;` | *Работает*. |
| F031-03 | Инструкция GRANT | | Ошибка. Tarantool не поддерживает права, кроме как через NoSQL. |
| F031-04 | Инструкция ALTER TABLE: добавление столбца | `ALTER TABLE t7 ADD COLUMN t7_2 VARCHAR(1) DEFAULT 'q';` | Работает. Tarantool поддерживает *ALTER TABLE*, а в версии Tarantool 2.7 добавлена поддержка ADD COLUMN. |
| F031-13 | Инструкция DROP TABLE: предложение RESTRICT | `DROP TABLE t20 RESTRICT;` | Ошибка. Tarantool поддерживает *DROP TABLE*, но не предложение RESTRICT. |
| F031-16 | Инструкция DROP VIEW: предложение RESTRICT | `DROP VIEW v2 RESTRICT;` | Ошибка. Tarantool поддерживает *DROP VIEW*, но не предложение RESTRICT. |
| F031-19 | Инструкция REVOKE: предложение RESTRICT | | Ошибка. Tarantool не поддерживает права, кроме как через NoSQL. |

### F041, базовое объединение таблиц

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F041-01 | Операция inner join, но необязательно с ключевым словом INNER | `SELECT a.s1 FROM t7 a JOIN t7 b;` | *Работает*. |
| F041-02 | Ключевое слово INNER | `SELECT a.s1 FROM t7 a INNER JOIN t7 b;` | Работает. |
| F041-03 | LEFT OUTER JOIN | `SELECT t7.*,t22.* FROM t22 LEFT OUTER JOIN t7 ON (t22_1 = s1);` | Работает. |
| F041-04 | RIGHT OUTER JOIN | `SELECT t7.*,t22.* FROM t22 RIGHT OUTER JOIN t7 ON (t22_1 = s1);` | Ошибка — в синтаксисе. |
| F041-05 | Вложенные outer join | `SELECT t7.*,t22.* FROM t22 LEFT OUTER JOIN t7 ON (t22_1 = s1) LEFT OUTER JOIN t23;` | Работает. |
| F041-07 | Таблица, полученная в результате операции left или right outer join, может быть использована в inner join | `SELECT t7.* FROM (t22 LEFT OUTER JOIN t7 ON (t22_1 = s1)) j INNER JOIN t22 ON (j.t22_4 = t7.s1);` | Работает. |
| F041-08 | Все операторы сравнения поддерживаются | `SELECT * FROM t WHERE 0 = 1 OR 0 > 1 OR 0 < 1 OR 0 <> 1;` | *Работает*. |

### F051, базовая поддержка даты и времени

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F051-01 | Тип данных DATE (включая поддержку литерала DATE) | `CREATE TABLE dates (s1 DATE);` | Ошибка. Tarantool не поддерживает тип данных DATE. |
| F051-02 | Тип данных TIME (включая поддержку литерала TIME) | `CREATE TABLE times (s1 TIME DEFAULT TIME '1:2:3');` | Ошибка — в синтаксисе. |
| F051-03 | Тип данных TIMESTAMP (включая поддержку литерала TIMESTAMP) | `CREATE TABLE timestamps (s1 TIMESTAMP);` | Ошибка — в синтаксисе. |
| F051-04 | Предикат сравнения с типами данных DATE, TIME и TIMESTAMP | `SELECT * FROM dates WHERE s1 = s1;` | Ошибка. Типы данных даты и времени не поддерживаются. |
| F051-05 | Явное преобразование CAST между типами даты и времени и типами символьных строк | `SELECT cast(s1 AS VARCHAR(10)) FROM dates;` | Ошибка. Типы данных даты и времени не поддерживаются. |
| F051-06 | CURRENT_DATE | `SELECT current_date FROM t;` | Ошибка — в синтаксисе. |
| F051-07 | LOCALTIME | `SELECT localtime FROM t;` | Ошибка — в синтаксисе. |
| F051-08 | LOCALTIMESTAMP | `SELECT localtimestamp FROM t;` | Ошибка — в синтаксисе. |

### F081, UNION и EXCEPT в представлениях

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F081 | UNION и EXCEPT в представлениях | `CREATE VIEW vv AS SELECT * FROM t7 EXCEPT SELECT * * FROM t15;` | Работает. |

### F131, операции группировки

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F131-01 | Поддержка WHERE, GROUP BY и HAVING в запросах со сгруппированными представлениями | `CREATE VIEW vv2 AS SELECT * FROM vv GROUP BY s1;` | Работает. |
| F131-02 | Поддержка нескольких таблиц в запросах со сгруппированными представлениями | `CREATE VIEW vv3 AS SELECT * FROM vv2,t30;` | Работает. |
| F131-03 | Поддержка агрегатных функций в запросах со сгруппированными представлениями | `CREATE VIEW vv4 AS SELECT count(*) FROM vv2;` | Работает. |
| F131-04 | Подзапросы с предложениями GROUP BY и HAVING и сгруппированными представлениями | `CREATE VIEW vv5 AS SELECT count(*) FROM vv2 GROUP BY s1 HAVING count(*) > 0;` | Работает. |
| F131-05 | Инструкция SELECT, возвращающая одну строку, с GROUP BY и HAVING и сгруппированными представлениями | `SELECT count(*) FROM vv2 GROUP BY s1 HAVING count(*) > 0;` | Работает. |

### F181, поддержка нескольких модулей

Ошибка. В Tarantool отсутствуют модули.

### F201, функция CAST

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F201 | Функция CAST | `SELECT cast(s1 AS INT) FROM t;` | *Работает.* |

### F221, явно заданное значение по умолчанию

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F221 | Явно заданное значение по умолчанию | `UPDATE t SET s1 = DEFAULT;` | Ошибка — в синтаксисе. |

### F261, выражение CASE

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F261-01 | Простой CASE | `SELECT CASE WHEN 1 = 0 THEN 5 ELSE 7 END FROM t;` | Работает. |
| F261-02 | CASE с поиском | `SELECT CASE 1 WHEN 0 THEN 5 ELSE 7 END FROM t;` | Работает. |
| F261-03 | NULLIF | `SELECT nullif(s1,7) FROM t;` | *Работает* |
| F261-04 | COALESCE | `SELECT coalesce(s1,7) FROM t;` | *Работает*. |

### F311, инструкция определения схемы

| Код возможности | Возможность | Результаты тестов |
|---|---|---|
| F311-01 | CREATE SCHEMA | Ошибка. В Tarantool отсутствуют схемы и базы данных. |
| F311-02 | CREATE TABLE для хранимых таблиц БД | Ошибка. В Tarantool нет CREATE TABLE в CREATE SCHEMA. |
| F311-03 | CREATE VIEW | Ошибка. В Tarantool нет CREATE VIEW в CREATE SCHEMA. |
| F311-04 | CREATE VIEW: WITH CHECK OPTION | Ошибка. В Tarantool нет CREATE VIEW в CREATE SCHEMA. |
| F311-05 | Инструкция GRANT | Ошибка. В Tarantool в составе CREATE SCHEMA нет GRANT. |

### F*, прочее

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| F471 | Скалярные значения подзапросов | `SELECT s1 FROM t WHERE s1 = (SELECT count(*) FROM t);` | Работает. |
| F481 | Расширенный предикат NULL | `SELECT * FROM t WHERE row(s1,s1) IS NOT NULL;` | Ошибка — в синтаксисе. |
| F812 | Базовые флаги | | Ошибка. Tarantool не поддерживает флаги. |

### S011, пользовательские типы

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| S011 | Пользовательские типы | `CREATE TYPE x AS FLOAT;` | Ошибка. Tarantool не поддерживает пользовательские типы. |

### T321, базовые процедуры, вызываемые SQL

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| T321-01 | Пользовательские функции без переопределения | `CREATE FUNCTION f() RETURNS INT RETURN 5;` | Ошибка. Пользовательские функции для SQL создаются с помощью *Lua* с другим синтаксисом. |
| T321-02 | Пользовательские процедуры без переопределения | `CREATE PROCEDURE p() BEGIN END;` | Ошибка. Пользовательские функции для SQL создаются с помощью *Lua* с другим синтаксисом. |
| T321-03 | Вызов функций | `SELECT f(1) FROM t;` | Работает. Tarantool может вызывать пользовательские Lua-функции. |
| T321-04 | Инструкция CALL | `CALL p();` | Ошибка. Tarantool не поддерживает инструкции CALL. |
| T321-05 | Инструкция RETURN | `CREATE FUNCTION f() RETURNS INT RETURN 5;` | Ошибка. Tarantool не поддерживает инструкции RETURN. |

### T*, прочее

| Код возможности | Возможность | Пример | Результаты тестов |
|---|---|---|---|
| T631 | Предикат IN с одним элементом списка | `SELECT * FROM t WHERE 1 IN (1);` | Работает. |

Количество элементов с отметкой «Ошибка»: 67.

Количество элементов с отметкой «Работает»: 79.

## 9.5 Справочник по встроенным модулям

В данном справочнике рассматриваются встроенные Lua-модули Tarantool.

---

**Примечание:** Некоторые функции в данных модулях представляют собой аналоги функций из стандартных Lua-библиотек. Для достижения наилучшего результата мы рекомендуем использовать функции из встроенных модулей Tarantool.

---

### 9.5.1 Модуль box

Помимо выполнения фрагментов кода на Lua или определения собственных функций, с помощью модуля `box` и вложенных модулей можно использовать функции хранилища Tarantool.

Каждый вложенный модуль включает в себя одну или более Lua-функций. Несколько вложенных модулей включают в себя элементы класса, а также функции. Функции обеспечивают определение данных (create alter drop), управление данными (insert delete update upsert select replace) и просмотр состояния (просмотр содержимого спейсов, получение доступа к конфигурации сервера).

Чтобы найти ошибки, которые могут выдать вложенные модули `box`, используйте *pcall*.

Содержимое модуля `box` можно просмотреть во время исполнения кода с помощью команды `box` без аргументов. Модуль `box` включает в себя следующее:

#### Вложенный модуль box.backup

Модуль `box.backup` содержит две функции, которые помогают при работе с *резервированным копированием*.

Ниже приведен перечень всех функций модуля `box.backup`.

| Имя | Использование |
|---|---|
| *box.backup.start()* | попросить сервер приостановить активности перед удалением устаревших резервных копий |
| *box.backup.stop()* | проинформировать сервер, что можно вернуться к работе |

#### box.backup.start()

`backup.start(`$\left[n\right]$`)`

Оповещает сервер о том, что следует приостановить все активности, связанные с удалением устаревших резервных копий.

Чтобы гарантировать возможность скопировать эти файлы, Tarantool не станет их удалять. При этом он не переходит в режим read-only, и создание контрольных точек делается по расписанию, как обычно.

**Параметры**

- `n` (`number`) – необязательный аргумент, начиная с Tarantool 1.10.1, который указывает нужную контрольную точку относительно самой свежей точки. Например, `n = 0` означает “резервная копия будет создана на основе самой свежей контрольной точки”, `n = 1` означает «резервная копия будет создана на основе

контрольной точки, которая была создана перед самой свежей точкой», и т.д. По умолчанию n = 0.

**Возвращает:** таблицу с именами снапшотов и файлов vinyl, которые нужно скопировать

**Пример:**

```
tarantool> box.backup.start()
---
- - ./00000000000000000015.snap
  - ./00000000000000000000.vylog
  - ./513/0/00000000000000000002.index
  - ./513/0/00000000000000000002.run
...
```

### box.backup.stop()

backup.stop()

Оповещает сервер о том, что можно вернуться к работе в обычном режиме.

### Вложенный модуль box.cfg

Вложенный модуль `box.cfg` предназначен для системных администраторов, чтобы указать все *параметры конфигурации сервера*.

Чтобы посмотреть текущую конфигурацию, вызовите `box.cfg` без скобок:

```
tarantool> box.cfg
---
- checkpoint_count: 2
  too_long_threshold: 0.5
  slab_alloc_factor: 1.05
  memtx_max_tuple_size: 1048576
  background: false
  <...>
...
```

Для установки конкретных параметров используйте следующий синтаксис: `box.cfg{key = value [, key = value ...] }` (далее для краткости `box.cfg{...}`). Например:

```
tarantool> box.cfg{listen = 3301}
```

Параметры, не указанные в явном вызове `box.cfg{...}`, будут установлены в *значения по умолчанию*.

Если ввести `box.cfg{}` без параметров, Tarantool применит настройки по умолчанию:

```
tarantool> box.cfg{}
tarantool> box.cfg -- sorted in the alphabetic order
---
- background                  = false
  checkpoint_count            = 2
  checkpoint_interval         = 3600
  checkpoint_wal_threshold    = 1000000000000000000
  coredump                    = false
```

(continues on next page)

(продолжение с предыдущей страницы)

```
custom_proc_title           = nil
feedback_enabled            = true
feedback_host               = 'https://feedback.tarantool.io'
feedback_interval           = 3600
force_recovery              = false
hot_standby                 = false
instance_uuid               = nil -- generated automatically
io_collect_interval         = nil
iproto_threads              = 1
listen                      = nil
log                         = nil
log_format                  = plain
log_level                   = 5
log_nonblock                = true
memtx_dir                   = '.'
memtx_max_tuple_size        = 1024 * 1024
memtx_memory                = 256 * 1024 *1024
memtx_min_tuple_size        = 16
net_msg_max                 = 768
pid_file                    = nil
readahead                   = 16320
read_only                   = false
replicaset_uuid             = nil -- generated automatically
replication                 = nil
replication_anon            = false
replication_connect_timeout = 30
replication_skip_conflict   = false
replication_sync_lag        = 10
replication_sync_timeout    = 300
replication_timeout         = 1
slab_alloc_factor           = 1.05
snap_io_rate_limit          = nil
sql_cache_size              = 5242880
strip_core                  = true
too_long_threshold          = 0.5
username                    = nil
vinyl_bloom_fpr             = 0.05
vinyl_cache                 = 128 * 1024 * 1024
vinyl_dir                   = '.'
vinyl_max_tuple_size        = 1024 * 1024* 1024 * 1024
vinyl_memory                = 128 * 1024 * 1024
vinyl_page_size             = 8 * 1024
vinyl_range_size            = nil
vinyl_read_threads          = 1
vinyl_run_count_per_level   = 2
vinyl_run_size_ratio        = 3.5
vinyl_timeout               = 60
vinyl_write_threads         = 4
wal_dir                     = '.'
wal_dir_rescan_delay        = 2
wal_max_size                = 256 * 1024 * 1024
wal_mode                    = 'write'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
worker_pool_threads          = 4
work_dir                     = nil
```

Первый вызов `box.cfg{...}` (с параметрами или без них) запускает модуль базы данных Tarantool под названием *box*.

Команда `box.cfg{...}` также перезагружает *файлы с данными длительного хранения* в оперативную память при перезапуске после получения данных.

### Вложенный модуль box.ctl

The `wait_ro` (wait until read-only) and `wait_rw` (wait until read-write) functions are useful during server initialization. To see whether a function is already in read-only or read-write mode, check *box.info.ro*.

Для *box.once()* есть особое предназначение. Например, при инициализации реплика может вызвать функцию `box.once()`, пока сервер все еще находится в режиме только для чтения, и не сможет применить изменения однократно до окончательной инициализации реплики. Это может привести к конфликту между мастером и репликой, если мастер находится в режиме чтения и записи, а реплика доступна только для чтения. Ожидание условия «read only mode = false» (режим только для чтения отключен) решает эту проблему.

Ниже приведен перечень всех функций модуля `box.ctl`.

| Имя | Использование |
|---|---|
| *box.ctl.wait_ro()* | Дождаться, пока не будет выполнено `box.info.ro` |
| *box.ctl.wait_rw()* | Дождаться, пока не перестанет соблюдаться `box.info.ro` |
| *box.ctl.on_schema_init()* | Создать «schema_init триггер» |
| *box.ctl.on_shutdown()* | Создать «shutdown триггер» |
| *box.ctl.on_recovery_state()* | Create a trigger executed on different stages of a node recovery or initial configuration |
| *box.ctl.on_election()* | Create a *trigger* executed every time the current state of a replica set node in regard to *leader election* changes |
| *box.ctl.set_on_shutdown_timeout()* | Set a timeout in seconds for the `on_shutdown` trigger |
| *box.ctl.is_recovery_finished()* | Check if recovery has finished |
| *box.ctl.promote()* | Wait, then choose replication leader |
| *box.ctl.demote()* | Revoke the leader role from the instance |
| *box.ctl.make_bootstrap_leader()* | Make the instance a bootstrap leader of a replica set |

### box.ctl.wait_ro()

`box.ctl.wait_ro`([*timeout*])

Дождаться, пока не будет выполнено `box.info.ro`.

**Параметры**

- `timeout` (`number`) – максимальное количество секунд ожидания

**возвращает** нулевое значение nil или ошибка, которая может возникнуть из-за превышения времени ожидания или прерывания работы файбера

**Пример:**

```
tarantool> box.info().ro
---
- false
...

tarantool> n = box.ctl.wait_ro(0.1)
---
- error: timed out
...
```

### box.ctl.wait_rw()

box.ctl.wait_rw([*timeout*])
Дождаться, пока не перестанет соблюдаться box.info.ro.

**Параметры**

- timeout (number) – максимальное количество секунд ожидания

**возвращает** нулевое значение nil или ошибка, которая может возникнуть из-за превышения времени ожидания или прерывания работы файбера

**Пример:**

```
tarantool> box.ctl.wait_rw(0.1)
---
...
```

### box.ctl.on_schema_init()

box.ctl.on_schema_init(*trigger-function*[, *old-trigger-function*])
Создать *триггер* schema_init. Функция триггера будет выполнена, когда *box.cfg{}* произойдет в первый раз. То есть триггер schema_init вызывается до начала конфигурирования и восстановления сервера, и поэтому box.ctl.on_schema_init должен быть вызван до вызова box.cfg.

**Параметры**

- trigger-function (function) – функция, в которой будет триггер
- old-trigger-function (function) – существующая функция с триггером, которую заменит новая

**возвращает** nil или указатель функции

Если указаны параметры (nil, old-trigger-function), старый триггер будет удален.

Обычно используется следующее: сделать триггерную функцию schema_init, которая создает триггерную функцию before_replace на системном спейсе. Таким образом, поскольку системные спейсы создаются при старте сервера, триггеры before_replace будут активированы для каждого кортежа в каждом системном спейсе. Например, такой триггер может изменить механизм хранения заданного спейса, или сделать заданный спейс *локальной репликой* во время загрузки реплики. Выполнение такого изменения после box.cfg не является надежным, поскольку другие подключения могут использовать базу данных до внесения изменения.

Подробная информация о характеристиках триггера находится в разделе *Триггеры*.

**Пример:**

Предположим, что до того, как сервер будет полностью готов к подключениям, вы хотите убедиться, что движком `space_name` является vinyl. Поэтому вы хотите сделать триггер, который будет активирован при вставке кортежа в системный спейс `_space`. В этом случае может получиться мастер, который имеет space-name с `engine='memtx'` и реплику, которая имеет space_name с `engine='vinyl'`, с тем же самым содержимым.

```
function function_for_before_replace(old, new)
  if old == nil and new ~= nil and new[3] == 'space_name' and new[4] ~= 'vinyl' then
    return new:update{{'=', 4, 'vinyl'}}
  end
end

box.ctl.on_schema_init(function()
  box.space._space:before_replace(function_for_before_replace)
end)

box.cfg{replication='master_uri', ...}
```

### box.ctl.on_shutdown()

box.ctl.on_shutdown(*trigger-function*[, *old-trigger-function*])

Создать «*триггер* выключения». Триггер-функция будет выполняться всякий раз, когда происходит *os.exit()*, или когда сервер выключается после получения сигнала SIGTERM или SIGINT или SIGHUP (но не после сигнала SIGSEGV или SIGABORT или любого другого сигнала, вызывающего немедленное завершение программы).

**Параметры**

- `trigger-function` (`function`) – функция, в которой будет триггер
- `old-trigger-function` (`function`) – существующая функция с триггером, которую заменит новая

**возвращает** nil или указатель функции

Если указаны параметры (nil, old-trigger-function), старый триггер будет удален.

Если вы хотите установить таймаут для этого триггера, используйте функцию *set_on_shutdown_timeout*.

### box.ctl.on_recovery_state()

box.ctl.on_recovery_state(*trigger-function*)

**Since:** *2.11.0*

Create a *trigger* executed on different stages of a node *recovery* or initial configuration. Note that you need to set the `box.ctl.on_recovery_state` trigger before the initial *box.cfg* call.

**Параметры**

- `trigger-function` (`function`) – a trigger function

**Return** `nil` or a function pointer

A registered trigger function is run on each of the supported recovery state and receives the state name as a parameter:

- `snapshot_recovered`: the node has recovered the snapshot files.
- `wal_recovered`: the node has recovered the WAL files.
- `indexes_built`: the node has built secondary indexes for memtx spaces. This stage might come before any actual data is recovered. This means that the indexes are available right after the first tuple is recovered.
- `synced`: the node has synced with enough remote peers. This means that the node changes the state from *orphan* to `running`.

All these states are passed during the initial `box.cfg` call when recovering from the snapshot and WAL files. Note that the `synced` state might be reached after the initial `box.cfg` call finishes. For example, if *replication_sync_timeout* is set to 0, the node finishes `box.cfg` without reaching `synced` and stays `orphan`. Once the node is synced with enough remote peers, the `synced` state is reached.

---

**Примечание:** When bootstrapping a fresh cluster with no data, all the instances in this cluster execute triggers on the same stages for consistency. For example, `snapshot_recovered` and `wal_recovered` run when the node finishes a cluster's bootstrap or finishes joining to an existing cluster.

---

**Example:**

The example below shows how to *log* a specified message when each state is reached.

```
local log = require('log')
local log_recovery_state = function(state)
    log.info(state .. ' state reached')
end
box.ctl.on_recovery_state(log_recovery_state)
```

### box.ctl.on_election()

`box.ctl.on_election`(*trigger-function*)

**Since:** *2.10.0*

Create a *trigger* executed every time the current state of a replica set node in regard to *leader election* changes. The current state is available in the *box.info.election* table.

The trigger doesn't accept any parameters. You can see the changes in `box.info.election` and *box.info.synchro*.

**Параметры**

- `trigger-function` (`function`) – a trigger function

**Return** `nil` or a function pointer

### box.ctl.set_on_shutdown_timeout()

box.ctl.set_on_shutdown_timeout([*timeout*])
Set a timeout for the *on_shutdown* trigger. If the timeout has expired, the server stops immediately regardless of whether any `on_shutdown` triggers are left unexecuted.

**Параметры**

- `timeout` (`double`) – time to wait for the trigger to be completed. The default value is 3 seconds.

**Return** nil

### box.ctl.is_recovery_finished()

box.ctl.is_recovery_finished()
Since version *2.5.3*.

Check whether the *recovery process* has finished. Until it has finished, space changes such as `insert` or `update` are not possible.

**возвращает** `true` if recovery has finished, otherwise `false`

**возвращаемое значение** boolean (логический)

### box.ctl.promote()

promote()
Since version *2.6.2*. Renamed in release *2.6.3*.

Wait, then choose new replication leader.

For *synchronous transactions* it is possible that a new leader will be chosen but the transactions of the old leader have not been completed. Therefore to finalize the transaction, the function `box.ctl.promote()` should be called, as mentioned in the notes for *leader election*. The old name for this function is `box.ctl.clear_synchro_queue()`.

The *election state* should change to `leader`.

Parameters: none

**возвращает** nil или указатель функции

### box.ctl.demote()

demote()
Since version *2.10.0*.

Revoke the leader role from the instance.

On *synchronous transaction queue owner*, the function works in the following way:

- If *box.cfg.election_mode* is `off`, the function writes a `DEMOTE` request to WAL. The `DEMOTE` request clears the ownership of the synchronous transaction queue, while the `PROMOTE` request assigns it to a new instance.
- If *box.cfg.election_mode* is enabled in any mode, then the function makes the instance start a new term and give up the leader role.

On instances that are not queue owners, the function does nothing and returns immediately.

Parameters: none

**Return** nil

### box.ctl.make_bootstrap_leader()

make_bootstrap_leader()

**Since:** *3.0.0*.

Make the instance a bootstrap leader of a replica set.

To be able to make the instance a bootstrap leader manually, the *replication.bootstrap_strategy* configuration option should be set to `supervised`. In this case, the instances do not choose a bootstrap leader automatically but wait for it to be appointed manually. Configuration fails if no bootstrap leader is appointed during a *replication.connect_timeout*.

---

**Примечание:** When a new instance joins a replica set configured with the `supervised` bootstrap strategy, this instance doesn't choose the bootstrap leader automatically but joins to the instance on which `box.ctl.make_bootstrap_leader()` was executed last time.

---

## Вложенный модуль box.error

The `box.error` submodule can be used to work with errors in your application. For example, you can get the information about the last error raised by Tarantool or raise custom errors manually.

The difference between raising an error using `box.error` and a Lua's built-in error function is that when the error reaches the client, its error code is preserved. In contrast, a Lua error would always be presented to the client as `ER_PROC_LUA`.

---

**Примечание:** To learn how to handle errors in your application, see the *Handling errors* section.

---

### Creating an error

You can create an error object using the *box.error.new()* function. The created object can be passed to *box.error()* to raise the error. You can also raise the error using *error_object:raise()*.

The example below shows how to create and raise the error with the specified code and reason.

```
local custom_error = box.error.new({ code = 500,
                                     reason = 'Internal server error' })

box.error(custom_error)
--[[
---
- error: Internal server error
...
--]]
```

`box.error.new()` provides different overloads for creating an error object with different parameters. These overloads are similar to the `box.error()` overloads described in the *next section*.

### Raising an error

To raise an error, call the *box.error()* function. This function can accept the specified error parameters or an error object *created using box.error.new()*. In both cases, you can use `box.error()` to raise the following error types:

- A custom error with the specified reason, code, and type.
- A predefined Tarantool error.

### Custom error

The following `box.error()` overloads are available for raising a custom error:

- *box.error({ reason = string[, code = number, type = string] })* accepts a Lua table containing the error reason, code, and type.
- *box.error(type, reason[, args])* accepts the error type, its reason, and optional arguments passed to a reason's string.

---

**Примечание:** The same overloads are available for *box.error.new()*.

---

#### box.error({ reason = string[, ...] })

In the example below, `box.error()` accepts a Lua table with the specified error code and reason:

```
box.error { code = 500,
            reason = 'Custom server error' }
--[[
---
- error: Custom server error
...
--]]
```

The next example shows how to specify a custom error type:

```
box.error { code = 500,
            reason = 'Internal server error',
            type = 'CustomInternalError' }
--[[
---
- error: Internal server error
...
--]]
```

When a custom type is specified, it is returned in the *error_object.type* attribute. When it is not specified, `error_object.type` returns one of the built-in errors, such as `ClientError` or `OutOfMemory`.

#### box.error(type, reason[, ...])

This example shows how to raise an error with the type and reason specified in the `box.error()` arguments:

```
box.error('CustomConnectionError', 'cannot connect to the given port')
--[[
---
- error: cannot connect to the given port
...
--]]
```

You can also use a format string to compose an error reason:

```
box.error('CustomConnectionError', '%s cannot connect to the port %u', 'client', 8080)
--[[
---
- error: client cannot connect to the port 8080
...
--]]
```

#### Tarantool error

The *box.error(code[, ...])* overload raises a predefined *Tarantool error* specified by its identifier. The error code defines the error message format and the number of required arguments. In the example below, no arguments are passed for the `box.error.READONLY` error code:

```
box.error(box.error.READONLY)
--[[
---
- error: Can't modify data on a read-only instance
...
--]]
```

For the `box.error.NO_SUCH_USER` error code, you need to pass one argument:

```
box.error(box.error.NO_SUCH_USER, 'John')
--[[
---
- error: User 'John' is not found
...
--]]
```

`box.error.CREATE_SPACE` requires two arguments:

```
box.error(box.error.CREATE_SPACE, 'my_space', 'the space already exists')
--[[
---
- error: 'Failed to create space ''my_space'': the space already exists'
...
--]]
```

### Getting the last error

To get the last raised error, call *box.error.last()*:

```
box.error.last()
--[[
---
- error: Internal server error
...
--]]
```

### Obtaining error details

To get error details, call the *error_object.unpack()*. Error details may include an error code, type, message, and trace.

```
box.error.last():unpack()
--[[
---
- code: 500
  base_type: CustomError
  type: CustomInternalError
  custom_type: CustomInternalError
  message: Internal server error
  trace:
  - file: '[string "custom_error = box.error.new({ code = 500,..."]'
    line: 1
...
--]]
```

### Setting the last error

You can set the last error explicitly by calling *box.error.set()*:

```
-- Create two errors --
local error1 = box.error.new({ code = 500, reason = 'Custom error 1' })
local error2 = box.error.new({ code = 505, reason = 'Custom error 2' })

-- Raise the first error --
box.error(error1)
--[[
---
- error: Custom error 1
...
--]]

-- Get the last error --
box.error.last()
--[[
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- Custom error 1
...
--]]

-- Set the second error as the last error --
box.error.set(error2)
--[[
---
...
--]]

-- Get the last error --
box.error.last()
--[[
---
- Custom error 2
...
--]]
```

### Error lists

*error_object* provides the API for organizing errors into lists. To set and get the previous error, use the *error_object:set_prev()* method and *error_object.prev* attribute.

```
local base_server_error = box.error.new({ code = 500,
                                          reason = 'Base server error',
                                          type = 'BaseServerError' })
local storage_server_error = box.error.new({ code = 507,
                                             reason = 'Not enough storage',
                                             type = 'StorageServerError' })

base_server_error:set_prev(storage_server_error)
--[[
---
...
--]]

box.error(base_server_error)
--[[
---
- error: Base server error
...
--]]

box.error.last().prev:unpack()
--[[
---
- code: 507
  base_type: CustomError
  type: StorageServerError
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  custom_type: StorageServerError
  message: Not enough storage
  trace:
  - file: '[string "storage_server_error = box.error.new({ code =..."]'
    line: 1
...
--]]
```

Cycles are not allowed for error lists:

```
storage_server_error:set_prev(base_server_error)
--[[
---
- error: 'builtin/error.lua:120: Cycles are not allowed'
...
--]]
```

Setting the previous error does not erase its own previous members:

```
-- e1 -> e2 -> e3 -> e4
e1:set_prev(e2)
e2:set_prev(e3)
e3:set_prev(e4)
e2:set_prev(e5)
-- Now there are two lists: e1 -> e2 -> e5 and e3 -> e4
```

IPROTO also supports stacked diagnostics. See details in *MessagePack extensions – The ERROR type*.

### Clearing errors

To clear the errors, call *box.error.clear()*.

```
box.error.clear()
--[[
---
...
--]]
box.error.last()
--[[
---
- null
...
--]]
```

### API Reference

Below is a list of `box.error` functions and related objects.

| Имя | Назначение |
|---|---|
| *box.error()* | Raise the last error or the error defined by the specified parameters |
| *box.error.last()* | Get the last raised error |
| *box.error.clear()* | Clear the errors |
| *box.error.new()* | Create the error but do not raise it |
| *box.error.set()* | Set the specified error as the last system error explicitly |
| *error_object* | An object that defines an error |

### box.error()

`box.error()`
Raise the last error.

**See also:** *box.error.last()*

`box.error(`*error_object*`)`
Raise the error defined by *error_object*.

**Параметры**

- `error_object` (*error_object*) – an error object

**Example**

```
local custom_error = box.error.new({ code = 500,
                                     reason = 'Internal server error' })

box.error(custom_error)
--[[
---
- error: Internal server error
...
--]]
```

`box.error(`*{ reason = string*[*, code = number, type = string] }*`)`
Raise the error defined by the specified parameters.

**Параметры**

- `reason` (*string*) – an error description
- `code` (`integer`) – (optional) a numeric code for this error
- `type` (*string*) – (optional) an error type

**Example 1**

```
box.error { code = 500,
            reason = 'Custom server error' }
--[[
---
- error: Custom server error
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
--]]
```

**Example 2: custom type**

```
box.error { code = 500,
            reason = 'Internal server error',
            type = 'CustomInternalError' }
--[[
---
- error: Internal server error
...
--]]
```

box.error(*type*, *reason*[, ...])
Raise the error defined by the specified type and description.

**Параметры**

- type (*string*) – an error type
- reason (*string*) – an error description
- ... – description arguments

**Example 1: without arguments**

```
box.error('CustomConnectionError', 'cannot connect to the given port')
--[[
---
- error: cannot connect to the given port
...
--]]
```

**Example 2: with arguments**

```
box.error('CustomConnectionError', '%s cannot connect to the port %u', 'client',␣
↪8080)
--[[
---
- error: client cannot connect to the port 8080
...
--]]
```

box.error(*code*[, ...])
Raise a predefined *Tarantool error* specified by its identifier. You can see all Tarantool errors in the errcode.h file.

**Параметры**

- code (number) – a pre-defined error identifier; Lua constants that correspond to those Tarantool errors are defined as members of box.error, for example, box.error.NO_SUCH_USER == 45
- ... – description arguments

**Example 1: no arguments**

```
box.error(box.error.READONLY)
--[[
---
- error: Can't modify data on a read-only instance
...
--]]
```

**Example 2: one argument**

```
box.error(box.error.NO_SUCH_USER, 'John')
--[[
---
- error: User 'John' is not found
...
--]]
```

**Example 3: two arguments**

```
box.error(box.error.CREATE_SPACE, 'my_space', 'the space already exists')
--[[
---
- error: 'Failed to create space ''my_space'': the space already exists'
...
--]]
```

### box.error.last()

box.error.last()
Get the last raised error.

**Return** an *error_object* representing the last error

**Example**

```
box.error.last()
--[[
---
- error: Internal server error
...
--]]
```

**See also:** *error_object:unpack()*

### box.error.clear()

box.error.clear()
Clear the errors.

**Example**

```
box.error.clear()
--[[
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...
--]]
box.error.last()
--[[
---
- null
...
--]]
```

### box.error.new()

box.error.new(*{ reason = string*[*, code = number, type = string] }*])

Create an error object with the specified parameters.

**Параметры**

- reason (*string*) – an error description
- code (integer) – (optional) a numeric code for this error
- type (*string*) – (optional) an error type

**Example 1**

```
local custom_error = box.error.new({ code = 500,
                                     reason = 'Internal server error' })

box.error(custom_error)
--[[
---
- error: Internal server error
...
--]]
```

**Example 2: custom type**

```
local custom_error = box.error.new({ code = 500,
                                     reason = 'Internal server error',
                                     type = 'CustomInternalError' })

box.error(custom_error)
--[[
---
- error: Internal server error
...
--]]
```

box.error.new(*type, reason*[*, ...*])

Create an error object with the specified type and description.

**Параметры**

- type (*string*) – an error type

- reason (*string*) – an error description
- ... – description arguments

**Example**

```
local custom_error = box.error.new('CustomInternalError', 'Internal server error')

box.error(custom_error)
--[[
---
- error: Internal server error
...
--]]
```

box.error.new(*code*[, ...])

Create a predefined *Tarantool error* specified by its identifier. You can see all Tarantool errors in the errcode.h file.

**Параметры**

- code (number) – a pre-defined error identifier; Lua constants that correspond to those Tarantool errors are defined as members of box.error, for example, box.error.NO_SUCH_USER == 45
- ... – description arguments

**Example 1: one argument**

```
local custom_error = box.error.new(box.error.NO_SUCH_USER, 'John')

box.error(custom_error)
--[[
---
- error: User 'John' is not found
...
--]]
```

**Example 2: two arguments**

```
local custom_error = box.error.new(box.error.CREATE_SPACE, 'my_space', 'the space␣
↪already exists')

box.error(custom_error)
--[[
---
- error: 'Failed to create space ''my_space'': the space already exists'
...
--]]
```

### box.error.set()

box.error.set(*error_object*)

**Since:** *2.4.1*

Set the specified error as the last system error explicitly. This error is returned by *box.error.last()*.

**Параметры**

- error_object (*error_object*) – an error object

**Example**

```
-- Create two errors --
local error1 = box.error.new({ code = 500, reason = 'Custom error 1' })
local error2 = box.error.new({ code = 505, reason = 'Custom error 2' })

-- Raise the first error --
box.error(error1)
--[[
---
- error: Custom error 1
...
--]]

-- Get the last error --
box.error.last()
--[[
---
- Custom error 1
...
--]]

-- Set the second error as the last error --
box.error.set(error2)
--[[
---
...
--]]

-- Get the last error --
box.error.last()
--[[
---
- Custom error 2
...
--]]
```

### error_object

`object error_object`

An object that defines an error. `error_object` is returned by the following methods:

- *box.error.new()*
- *box.error.last()*

`error_object:unpack()`

Get error details that may include an error code, type, message, and trace.

**Example**

```
box.error.last():unpack()
--[[
---
- code: 500
  base_type: CustomError
  type: CustomInternalError
  custom_type: CustomInternalError
  message: Internal server error
  trace:
  - file: '[string "custom_error = box.error.new({ code = 500,..."]'
    line: 1
...
--]]
```

---

**Примечание:** Depending on the error type, error details may include other attributes, such as *errno* or *reason*.

---

`error_object:raise()`

Raise the current error.

**See also:** *Raising an error*

`error_object:set_prev(error_object)`

**Since:** *2.4.1*

Set the previous error for the current one.

**Параметры**

- `body` (*error_object*) – an error object

**See also:** *Error lists*

`error_object.prev`

**Since:** *2.4.1*

Get the previous error for the current one.

**Rtype** error_object

**See also:** *Error lists*

`error_object.code`

The error code. This attribute may return a *custom error* code or a *Tarantool error* code.

**Rtype** number

`error_object.type`
The error type.

**Rtype** string

**See also:** *Custom error*

`error_object.message`
The error message.

**Rtype** string

`error_object.trace`
The error trace.

**Rtype** table

`error_object.errno`
If the error is a system error (for example, a socket or file IO failure), returns a C standard error number.

**Rtype** number

`error_object.reason`
**Since:** *2.10.0*

Returns the *box.info.ro_reason* value at the moment of throwing the `box.error.READONLY` error.

The following values may be returned:

- `election` if the instance has *box.cfg.election_mode* set to a value other than `off` and this instance is not a leader. In this case, `error_object` may include the following attributes: `state`, `leader_id`, `leader_uuid`, and `term`.
- `synchro` if the synchronous queue has an owner that is not the given instance. This error usually happens if *synchronous replication* is used and another instance is called *box.ctl.promote()*. In this case, `error_object` may include the `queue_owner_id`, `queue_owner_uuid`, and `term` attributes.
- `config` if the *box.cfg.read_only* is set to `true`.
- `orphan` if the instance is in the *orphan* state.

---

**Примечание:** If multiple reasons are true at the same time, then only one is returned in the following order of preference: `election`, `synchro`, `config`, `orphan`.

---

**Rtype** string

`error_object.state`
**Since:** *2.10.0*

For the `box.error.READONLY` error, returns the current state of a replica set node in regards to leader election (see *box.info.election.state*). This attribute presents if the *error reason* is `election`.

**Rtype** string

`error_object.leader_id`
**Since:** *2.10.0*

For the `box.error.READONLY` error, returns a numeric identifier (*box.info.id*) of the replica set leader. This attribute may present if the *error reason* is `election`.

**Rtype** number

`error_object.leader_uuid`
**Since:** *2.10.0*

For the `box.error.READONLY` error, returns a globally unique identifier (*box.info.uuid*) of the replica set leader. This attribute may present if the *error reason* is `election`.

`error_object.queue_owner_id`
**Since:** *2.10.0*

For the `box.error.READONLY` error, returns a numeric identifier (*box.info.id*) of the synchronous queue owner. This attribute may present if the *error reason* is `synchro`.

**Rtype** number

`error_object.queue_owner_uuid`
**Since:** *2.10.0*

For the `box.error.READONLY` error, returns a globally unique identifier (*box.info.uuid*) of the synchronous queue owner. This attribute may present if the *error reason* is `synchro`.

`error_object.term`
**Since:** *2.10.0*

For the `box.error.READONLY` error, returns the current election term (see *box.info.election.term*). This attribute may present if the *error reason* is `election` or `synchro`.

### Вложенный модуль box.index

Вложенный модуль `box.index` обеспечивает доступ к схемам индекса и ключам индекса в режиме только для чтения. Индексы хранятся в массиве `box.space.`*`имя-спейса`*`.index` в каждом спейсе. Они предоставляют API для упорядоченной итерации по кортежам. Этот API представляет собой прямую привязку к соответствующим методам объектов типа `box.index` в движке базы данных.

Ниже приведен перечень всех функций и элементов модуля `box.index`.

| Имя | Использование |
|---|---|
| *Примеры для box.index* | Несколько полезных примеров |
| *space_object:create_index()* | Создание индекса |
| *index_object.unique* | Флаг, если индекс уникальный – true |
| *index_object.type* | Тип индекса |
| *index_object:parts* | Массив полей с ключами индекса |
| *index_object:pairs()* | Подготовка к итерации |
| *index_object:select()* | Выбор одного или более кортежей по индексу |
| *index_object:get()* | Выбор кортежа по индексу |
| *index_object:min()* | Поиск минимального значения в индексе |
| *index_object:max()* | Поиск максимального значения в индексе |
| *index_object:random()* | Поиск случайного значения в индексе |
| *index_object:count()* | Подсчет кортежей с совпадающим значением ключа |
| *index_object:update()* | Обновление кортежа |
| *index_object:delete()* | Удаление кортежа по ключу |
| *index_object:alter()* | Изменение индекса |
| *index_object:drop()* | Удаление индекса |
| *index_object:rename()* | Переименование индекса |
| *index_object:bsize()* | Подсчет байтов для индекса |
| *index_object:stat()* | Получение статистических данных по индексу |
| *index_object:compact()* | Удаление неиспользуемого пространства индекса |
| *index_object:tuple_pos()* | Return a tuple's position for an index |
| *index_object extensions* | Любая функция / метод, которые хочет добавить любой пользователь |

### Примеры для `box.index`

#### Пример использования функций box

Данный пример сработает на конфигурации из песочницы, описанной в предисловии, то есть создан спейс под названием tester с первичным числовым ключом. Функция в примере выполнит следующие действия:

- выбрать кортеж, значение ключа в котором равно 1000;
- выдать сообщение об ошибке, если такой кортеж уже существует и содержит 3 поля;
- **вставить или заменить кортеж следующими данными:**
  - поле [1] = 1000
  - поле [2] = UUID
  - поле [3] = количество секунд с 01.01.1970;
- получить поле [3] из того, что заменили;
- преобразовать значение из поля [3] в формат yyyy-mm-dd hh:mm:ss.ffff (год-месяц-день час:минута:секунда.десятитысячные доли секунды);
- вернуть преобразованное значение.

The function uses Tarantool box functions *box.space... select*, *box.space... replace*, *fiber.time*, *uuid.str*. The function uses Lua functions os.date() and string.sub().

```
function example()
  local a, b, c, table_of_selected_tuples, d
  local replaced_tuple, time_field
  local formatted_time_field
  local fiber = require('fiber')
  table_of_selected_tuples = box.space.tester:select{1000}
  if table_of_selected_tuples ~= nil then
    if table_of_selected_tuples[1] ~= nil then
      if #table_of_selected_tuples[1] == 3 then
        box.error({code=1, reason='This tuple already has 3 fields'})
      end
    end
  end
  replaced_tuple = box.space.tester:replace
    {1000,  require('uuid').str(), tostring(fiber.time())}
  time_field = tonumber(replaced_tuple[3])
  formatted_time_field = os.date("%Y-%m-%d %H:%M:%S", time_field)
  c = time_field % 1
  d = string.sub(c, 3, 6)
  formatted_time_field = formatted_time_field .. '.' .. d
  return formatted_time_field
end
```

... А вот что происходит, когда вызывается функция:

```
tarantool> box.space.tester:delete(1000)
---
- [1000, '264ee2da03634f24972be76c43808254', '1391037015.6809']
...
tarantool> example(1000)
---
- 2014-01-29 16:11:51.1582
...
tarantool> example(1000)
---
- error: 'This tuple already has 3 fields'
...
```

### Пример с заданным пользователем итератором

Здесь приведен пример того, как создать свой собственный итератор. Функция `paged_iter` представляет собой «функцию с итератором», что поймут только разработчики, которые ознакомились с разделом руководства по Lua Итераторы и замыкания. Она делает постраничную выборку, то есть возвращает 10 кортежей одновременно из таблицы под названием «t», первичный ключ которой определен с помощью `create_index('primary',{parts={1,'string'}})`.

```
function paged_iter(search_key, tuples_per_page)
  local iterator_string = "GE"
  return function ()
  local page = box.space.t.index[0]:select(search_key,
    {iterator = iterator_string, limit=tuples_per_page})
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  if #page == 0 then return nil end
  search_key = page[#page][1]
  iterator_string = "GT"
  return page
  end
end
```

Разработчикам, использующим `paged_iter`, необязательно знать, почему она работает, следует лишь понимать, что вызвав функцию в цикле, можно получать 10 кортежей за раз до тех пор, пока кортежи не кончатся.

В данном примере кортежи лишь выводятся по странице за раз. Но легко изменить функцию, например, путем передачи управления после каждой выборки или с помощью прерывания, если кортежи не будут соответствовать дополнительным критериям.

```
for page in paged_iter("X", 10) do
  print("New Page. Number Of Tuples = " .. #page)
  for i = 1, #page, 1 do
    print(page[i])
  end
end
```

### Пример использования `box.index` с индексом типа RTREE для поиска в пространственных данных

Этот вложенный модуль может использоваться для поиска в пространственных данных, если тип индекса – RTREE. Существуют операции для поиска *прямоугольников* (геометрические фигуры с 4 углами и 4 сторонами) и *параллелепипедов* (геометрические фигуры с количеством углов более 4 и количеством сторон более 4, которые иногда называются гиперпрямоугольниками). В данном руководстве используется термин *прямоугольник-или-параллелепипед* для всего класса объектов, который включает в себя прямоугольники и параллелепипеды. Примерами иллюстрируются только прямоугольники.

Прямоугольники описаны в соответствии с координатами по оси X (горизонтальной оси) и оси Y (вертикальной оси) на сетке произвольного размера. Ниже представлен рисунок четырех прямоугольников на сетке с 11 горизонтальными точками и 11 вертикальными точками:

```
      X AXIS
          1   2   3   4   5   6   7   8   9   10  11
        1
        2  #-------+                                       <-Прямоугольник №1
Y AXIS  3  |       |
        4  +-------#
        5          #-----------------------+               <-Прямоугольник №2
        6          |                       |
        7          |   #---+               |               <-Прямоугольник №3
        8          |   |   |               |
        9          |   +---#               |
        10         +-----------------------#
        11                                 #               <-Прямоугольник №4
```

Прямоугольники определяются в соответствии со следующей схемой: {верхняя левая координата по оси X, верхняя левая координата по оси Y, нижняя правая координата по оси X, нижняя правая координата по оси Y} – или коротко: {x1,y1,x2,y2}. Таким образом, на рисунке ... Прямоугольник № 1 начинается в точке 1 по оси X и точке 2 по оси Y, а заканчивается в точке 3 по оси X и точке

4 по оси Y, поэтому его координаты будут следующие: {1,2,3,4}. Координаты Прямоугольника № 2: {3,5,9,10}. Координаты Прямоугольника № 3: {4,7,5,9}. И наконец, координаты Прямоугольника № 4: {10,11,10,11}. Прямоугольник № 4, на самом деле, является точкой, поскольку у него нулевая ширина и нулевая высота, так что его можно описать всего двумя числами: {10,11}.

Некоторые отношения между прямоугольниками могут быть описаны так: «Прямоугольник №1 является ближайшим соседом Прямоугольника №2», а «Прямоугольник №3 полностью находится внутри Прямоугольника №2».

Сейчас создадим спейс и добавим RTREE-индекс.

```
tarantool> s = box.schema.space.create('rectangles')
tarantool> i = s:create_index('primary', {
         >   type = 'HASH',
         >   parts = {1, 'unsigned'}
         > })
tarantool> r = s:create_index('rtree', {
         >   type = 'RTREE',
         >   unique = false,
         >   parts = {2, 'ARRAY'}
         > })
```

Поле №1 не имеет значения, мы создаем его лишь потому, что необходим первичный индекс. (RTREE-индексы не могут быть уникальными, поэтому не могут быть первичными индексами.) Второе поле должно быть массивом («array»), что означает, что его значения должны представлять собой точки {x,y} или прямоугольники {x1,y1,x2,y2}. Заполним таблицу, вставив два кортежа с координатами Прямоугольника №2 и Прямоугольника №4.

```
tarantool> s:insert{1, {3, 5, 9, 10}}
tarantool> s:insert{2, {10, 11}}
```

Затем, после описания типов RTREE-итераторов (*RTREE iterator types*), можно произвести поиск прямоугольников с помощью данных запросов:

```
tarantool> r:select({10, 11, 10, 11}, {iterator = 'EQ'})
---
- - [2, [10, 11]]
...
tarantool> r:select({4, 7, 5, 9}, {iterator = 'GT'})
---
- - [1, [3, 5, 9, 10]]
...
tarantool> r:select({1, 2, 3, 4}, {iterator = 'NEIGHBOR'})
---
- - [1, [3, 5, 9, 10]]
  - [2, [10, 11]]
...
```

Запрос №1 возвращает 1 кортеж, потому что точка {10,11} представляет собой то же, что и прямоугольник {10,11,10,11} («Прямоугольник №4» на рисунке). Запрос № 2 возвращает 1 кортеж, потому что прямоугольник {4,7,5,9}, который был «Прямоугольником №3» на рисунке находится полностью внутри {3,5,9,10}, что представляет собой Прямоугольник № 2. Запрос № 3 возвращает 2 кортежа, потому что итератор NEIGHBOR (сосед) всегда возвращает все кортежи, а первым найденным кортежем будет {3,5,9,10} («Прямоугольник №2» на рисунке), потому что он является ближайшим соседом {1,2,3,4} («Прямоугольник №1» на рисунке).

Теперь создадим спейс и индекс для кубоидов, которые представляют собой прямоугольники-или-параллелепипеды, у которых 6 углов и 6 сторон.

```
tarantool> s = box.schema.space.create('R')
tarantool> i = s:create_index('primary', {parts = {1, 'unsigned'}})
tarantool> r = s:create_index('S', {
         >   type = 'RTREE',
         >   unique = false,
         >   dimension = 3,
         >   parts = {2, 'ARRAY'}
         > })
```

Здесь задается дополнительный параметр `dimension=3`. По умолчанию, измерений 2, поэтому не было необходимости указывать данный параметр в примерах для прямоугольника. Максимальное количество измерений – 20. Что касается вставки и выборки, здесь будет 6 координат. Например:

```
tarantool> s:insert{1, {0, 3, 0, 3, 0, 3}}
tarantool> r:select({1, 2, 1, 2, 1, 2}, {iterator = box.index.GT})
```

Теперь создадим спейс и индекс для пространственных объектов с метрикой расстояния городских кварталов (метрика Манхэттена), которые представляют собой прямоугольники-или-параллелепипеды; соседи для них рассчитываются иным образом.

```
tarantool> s = box.schema.space.create('R')
tarantool> i = s:create_index('primary', {parts = {1, 'unsigned'}})
tarantool> r = s:create_index('S', {
         >   type = 'RTREE',
         >   unique = false,
         >   distance = 'manhattan',
         >   parts = {2, 'ARRAY'}
         > })
```

Здесь задается дополнительный параметр `distance='manhattan'`. По умолчанию, расстояние измеряется по Евклидовой метрике, что лучше всего подходит для измерений по прямой линии. Другой способ расчета расстояния по метрике Манхэттена („manhattan“), который больше подходит, если необходимо следовать линиям сетки, а не по прямой.

```
tarantool> s:insert{1, {0, 3, 0, 3}}
tarantool> r:select({1, 2, 1, 2}, {iterator = box.index.NEIGHBOR})
```

Другие примеры поиска в пространственных данных см. по ссылке *R tree index quick start and usage*.

### index_object.unique

`object index_object`

`index_object.unique`

`true` if the index is unique, `false` if the index is not unique.

**тип возвращаемого значения** boolean (логический)

**See also:** *index_opts.unique*

### index_object.type

```
object index_object
```

index_object.type
The *index type*.

**Rtype** string

**See also:** *index_opts.type*

### index_object:parts

```
object index_object
```

index_object.parts
The index's key parts.

**тип возвращаемого значения** таблица

**See also:** *index_opts.parts*

### index_object:pairs()

```
object index_object
```

index_object:pairs(*[key[, {iterator = iterator-type}]]*)
Search for a tuple or a set of tuples via the given index, and allow iterating over one tuple at a time. To search by the primary index in the specified space, use the *space_object:pairs()* method.

Параметр ***key*** (ключ) задает, что именно должно совпадать в индексе.

---

**Примечание:** ***key*** используется в поиске только первого совпадения. Не стоит ожидать, что все подобранные кортежи будут содержать этот ключ.

---

Параметр ***iterator*** (итератор) задает правило для совпадений и упорядочивания. Различные типы индексов поддерживают различные итераторы. Например, TREE-индекс поддерживает строгий порядок ключей и может вернуть все кортежи в порядке по возрастанию или по убыванию, начиная с указанного ключа. Однако другие типы индексов не поддерживают упорядочивание.

To understand consistency of tuples returned by an iterator, it's essential to know the principles of the Tarantool transaction processing subsystem. An iterator in Tarantool does not own a consistent read view. Instead, each procedure is granted exclusive access to all tuples and spaces until there is a «context switch»: which may happen due to *the implicit yield rules*, or by an explicit call to *fiber.yield*. When the execution flow returns to the yielded procedure, the data set could have changed significantly. Iteration, resumed after a yield point, does not preserve the read view, but continues with the new content of the database. The tutorial *Indexed pattern search* shows one way that iterators and yields can be used together.

For information about iterators" internal structures, see the «Lua Functional library» documentation.

**Параметры**

- `index_object` (*index_object*) – *ссылка на объект*.
- `key` (`scalar/table`) – value to be matched against the index key, which may be multi-part.
- `iterator` – as defined in tables below. The default iterator type is „EQ“.
- `after` – a tuple or the position of a tuple (*tuple_pos*) after which `pairs` starts the search. You can pass an empty string or *box.NULL* to this option to start the search from the first tuple.

**возвращает** The iterator, which can be used in a `for/end` loop or with totable().

**Возможные ошибки:**

- no such space
- wrong type
- selected iteration type is not supported for the index type
- key is not supported for the iteration type
- iterator position is invalid

**Факторы сложности** Размер индекса, тип индекса; количество кортежей, к которым получен доступ.

Значение искомого ключа может представлять собой число (например, `1234`), строку (например, `'abcd'`) или таблицу из чисел и строк (например, `{1234, 'abcd'}`). Каждая часть ключа будет сопоставляться с каждой частью ключа в индексе.

Найденные кортежи будут упорядочены по значению ключа в индексе или по хешу значения ключа, если тип индекса – „hash“. Если индекс не уникален, то дубликаты будут упорядочены во вторую очередь по первичному значению ключа. Порядок будет обратным, если тип итератора – „LT“, „LE“ или „REQ“.

**Типы итераторов для TREE-индексов**

| Тип итератора | Аргументы | Описание |
|---|---|---|
| box.index.EQ или „EQ“ | искомое значение | Оператором сравнения будет „==“ (равно). Если ключ индекса равен искомому значению, получим совпадение. Найденные кортежи упорядочены по возрастанию по ключу индекса. Этот тип используется по умолчанию. |
| box.index.REQ или „REQ“ | искомое значение | Совпадения находятся таким же образом, что и для `box.index.EQ`. Разница только в том, что найденные кортежи упорядочены по ключу индекса по убыванию, а не по возрастанию. |
| box.index.GT или „GT“ | искомое значение | Оператором сравнения будет „>“ (больше чем). Если ключ индекса больше, чем искомое значение, получим совпадение. Найденные кортежи упорядочены по возрастанию по ключу индекса. |
| box.index.GE или „GE“ | искомое значение | Оператором сравнения будет „>=“ (больше или равен). Если ключ индекса больше искомого значения или равен ему, получим совпадение. Найденные кортежи упорядочены по возрастанию по ключу индекса. |
| box.index.ALL или „ALL“ | искомое значение | Как для box.index.GE. |
| box.index.LT или „LT“ | искомое значение | Оператором сравнения будет „<“ (меньше чем). Если ключ индекса меньше искомого значения, получим совпадение. Найденные кортежи упорядочены по убыванию по ключу индекса. |
| box.index.LE или „LE“ | искомое значение | Оператором сравнения будет „<=“ (меньше или равен). Если ключ индекса меньше искомого значения или равен ему, получим совпадение. Найденные кортежи упорядочены по убыванию по ключу индекса. |

Неофициально можно сказать, что поиск с помощью TREE-индексов пользователи обычно считают интуитивно понятным при условии, что нет нулевых значений и отсутствующих частей. Формально же логика заключается в следующем. Ключ поиска состоит из нуля или более частей, например, {}, {1,2,3},{1,nil,3}. Ключ индекса состоит из одной или более частей, например, {1}, {1,2,3},{1,2,3}. Ключ поиска может содержать нулевое значение nil (но не msgpack.NULL, этот тип не будет правильным). Ключ индекса не может содержать nil или msgpack.NULL, хотя в последующих версиях правила работы Tarantool будут другие – поведение поиска с nil может измениться. Возможные итераторы: LT, LE, EQ, REQ, GE, GT. Считается, что ключ поиска соответствует ключу индекса, если следующие операторы, которые представляют собой псевдокод для операции сопоставления, возвращают TRUE.

```
If (number-of-search-key-parts > number-of-index-key-parts) return ERROR
If (number-of-search-key-parts == 0) return TRUE
for (i = 1; ; ++i)
{
  if (i > number-of-search-key-parts) OR (search-key-part[i] is nil)
  {
    if (iterator is LT or GT) return FALSE
    return TRUE
  }
  if (type of search-key-part[i] is not compatible with type of index-key-
↪part[i])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  {
    return ERROR
  }
  if (search-key-part[i] == index-key-part[i])
  {
    continue
  }
  if (search-key-part[i] > index-key-part[i])
  {
    if (iterator is EQ or REQ or LE or LT) return FALSE
    return TRUE
  }
  if (search-key-part[i] < index-key-part[i])
  {
    if (iterator is EQ or REQ or GE or GT) return FALSE
    return TRUE
  }
}
```

**Типы итераторов для HASH-индексов**

| Тип возвращаемого значения | Аргументы | Описание |
|---|---|---|
| box.index.ALL | нет | Все ключи индекса являются совпадениями. Найденные кортежи упорядочены по возрастанию по хешу ключа индекса, который будет выглядеть случайным. |
| box.index.EQ или „EQ“ | искомое значение | Оператором сравнения будет „==“ (равный). Если ключ индекса равен искомому значению, получим совпадение. Количество найденных кортежей будет 0 или 1. Этот тип используется по умолчанию. |

**Типы итераторов для BITSET-индексов**

| Тип возвращаемого значения | Аргументы | Описание |
|---|---|---|
| box.index.ALL или „ALL“ | нет | Все ключи индекса являются совпадениями. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.EQ или „EQ“ | значение bitset (битовое множество) | Если ключ индекса равен искомому значению, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. Этот тип используется по умолчанию. |
| box.index.BITS_ALL_SET | значение bitset (битовое множество) | Если все биты, которые равны 1 в битовом множестве, также равны 1 в ключе индекса, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.BITS_ANY_SET | значение bitset (битовое множество) | Если один из битов, которые равны 1 в битовом множестве, также равен 1 в ключе индекса, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.BITS_ALL_NOT_SET | значение bitset (битовое множество) | Если все биты, которые равны 1 в битовом множестве, равны 0 в ключе индекса, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. |

**Типы итераторов для RTREE-индексов**

| Тип возвращаемого значения | Аргументы | Описание |
|---|---|---|
| box.index.ALL или „ALL“ | нет | Все ключи являются совпадениями. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.EQ или „EQ“ | искомое значение | Если все точки прямоугольника-или-параллелепипеда, определенные искомым значением, совпадают с точками прямоугольника-или-параллелепипеда, определенного ключом индекса, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. «Прямоугольник-или-параллелепипед» означает «прямоугольник-или-параллелепипед, как описано в разделе о *RTREE*». Этот тип используется по умолчанию. |

continues on next page

Таблица 3 – продолжение с предыдущей страницы

| Тип возвращаемого значения | Аргументы | Описание |
|---|---|---|
| box.index.GT или „GT“ | искомое значение | Если все точки прямоугольника-или-параллелепипеда, определенные искомым значением, находятся в пределах прямоугольника-или-параллелепипеда, определенного ключом индекса, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.GE или „GE“ | искомое значение | Если все точки прямоугольника-или-параллелепипеда, определенные искомым значением, находятся в пределах прямоугольника-или-параллелепипеда, определенного ключом индекса, или рядом с ним, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.LT или „LT“ | искомое значение | Если все точки прямоугольника-или-параллелепипеда, определенные ключом индекса, находятся в пределах прямоугольника-или-параллелепипеда, определенного искомым значением, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.LE или „LE“ | искомое значение | Если все точки прямоугольника-или-параллелепипеда, определенные ключом индекса, находятся в пределах прямоугольника-или-параллелепипеда, определенного искомым значением, или рядом с ним, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.OVERLAPS или „OVERLAPS“ | искомое значение | Если некоторые точки прямоугольника-или-параллелепипеда, определенные искомым значением, находятся в пределах прямоугольника-или-параллелепипеда, определенного ключом индекса, получим совпадение. Найденные кортежи упорядочены по положению в спейсе. |
| box.index.NEIGHBOR или „NEIGHBOR“ | искомое значение | Если некоторые точки прямоугольника-или-параллелепипеда, определенные ключом, находятся в пределах, определенных ключом индекса, или рядом с ним, получим совпадение. Найденные кортежи упорядочены следующим образом: сначала ближайший сосед. |

**Examples:**

Below are few examples of using `pairs` with different parameters. To try out these examples, you need to bootstrap a Tarantool instance as described in *Using data operations*.

```
-- Insert test data --
tarantool> bands:insert{1, 'Roxette', 1986}
           bands:insert{2, 'Scorpions', 1965}
           bands:insert{3, 'Ace of Base', 1987}
           bands:insert{4, 'The Beatles', 1960}
           bands:insert{5, 'Pink Floyd', 1965}
           bands:insert{6, 'The Rolling Stones', 1962}
           bands:insert{7, 'The Doors', 1965}
           bands:insert{8, 'Nirvana', 1987}
           bands:insert{9, 'Led Zeppelin', 1968}
           bands:insert{10, 'Queen', 1970}
---
...

-- Select all tuples by the primary index --
tarantool> for _, tuple in bands.index.primary:pairs() do
               print(tuple)
           end
[1, 'Roxette', 1986]
[2, 'Scorpions', 1965]
[3, 'Ace of Base', 1987]
[4, 'The Beatles', 1960]
[5, 'Pink Floyd', 1965]
[6, 'The Rolling Stones', 1962]
[7, 'The Doors', 1965]
[8, 'Nirvana', 1987]
[9, 'Led Zeppelin', 1968]
[10, 'Queen', 1970]
---
...

-- Select all tuples whose secondary key values start with the specified string␣
↪--
tarantool> for _, tuple in bands.index.band:pairs("The", {iterator = "GE"}) do
             if (string.sub(tuple[2], 1, 3) ~= "The") then break end
             print(tuple)
           end
[4, 'The Beatles', 1960]
[7, 'The Doors', 1965]
[6, 'The Rolling Stones', 1962]
---
...

-- Select all tuples whose secondary key values are between 1965 and 1970 --
tarantool> for _, tuple in bands.index.year:pairs(1965, {iterator = "GE"}) do
             if (tuple[3] > 1970) then break end
             print(tuple)
           end
[2, 'Scorpions', 1965]
[5, 'Pink Floyd', 1965]
[7, 'The Doors', 1965]
[9, 'Led Zeppelin', 1968]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
[10, 'Queen', 1970]
---
...

-- Select all tuples after the specified tuple --
tarantool> for _, tuple in bands.index.primary:pairs({}, {after={7, 'The Doors',
↪ 1965}}) do
               print(tuple)
           end
[8, 'Nirvana', 1987]
[9, 'Led Zeppelin', 1968]
[10, 'Queen', 1970]
---
...
```

### index_object:select()

```
object index_object
```

index_object:select(*search-key*, *options*)
Search for a tuple or a set of tuples by the current index. To search by the primary index in the specified space, use the *space_object:select()* method.

**Параметры**

- `index_object` (*index_object*) – *ссылка на объект*.
- `key` (`scalar/table`) – a value to be matched against the index key, which may be multi-part.
- `options` (`table/nil`) – ни один, любой или все следующие параметры:
  - `iterator` – the *iterator type*. The default iterator type is „EQ“.
  - `limit` – the maximum number of tuples.
  - `offset` – the number of tuples to skip (use this parameter carefully when scanning *large data sets*).
  - `options.after` – a tuple or the position of a tuple (*tuple_pos*) after which `select` starts the search. You can pass an empty string or *box.NULL* to this option to start the search from the first tuple.
  - `options.fetch_pos` – if **true**, the `select` method returns the position of the last selected tuple as the second value.

  **Примечание:** The `after` and `fetch_pos` options are supported for the `TREE` *index* only.

**возвращает** This function might return one or two values:

- The tuples whose fields are equal to the fields of the passed key. If the number of passed fields is less than the number of fields in the current key, then only the passed fields are compared, so `select{1,2}` matches a tuple whose primary key is `{1,2,3}`.

- (Optionally) If `options.fetch_pos` is set to **true**, returns a base64-encoded string representing the position of the last selected tuple as the second value. If no tuples are fetched, returns `nil`.

**тип возвращаемого значения**

- массив кортежей
- (Optionally) string

**Предупреждение:** Use the `offset` option carefully when scanning large data sets as it linearly increases the number of scanned tuples and leads to a full space scan. Instead, you can use the `after` and `fetch_pos` options.

**Examples:**

Below are few examples of using `select` with different parameters. To try out these examples, you need to bootstrap a Tarantool database as described in *Using data operations*.

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.space.bands:insert { 5, 'Pink Floyd', 1965 }
box.space.bands:insert { 6, 'The Rolling Stones', 1962 }
box.space.bands:insert { 7, 'The Doors', 1965 }
box.space.bands:insert { 8, 'Nirvana', 1987 }
box.space.bands:insert { 9, 'Led Zeppelin', 1968 }
box.space.bands:insert { 10, 'Queen', 1970 }

-- Select a tuple by the specified primary key value --
select_primary = bands.index.primary:select { 1 }
--[[
---
- - [1, 'Roxette', 1986]
...
--]]

-- Select a tuple by the specified secondary key value --
select_secondary = bands.index.band:select { 'The Doors' }
--[[
---
- - [7, 'The Doors', 1965]
...
--]]

-- Select a tuple by the specified multi-part secondary key value --
select_multipart = bands.index.year_band:select { 1960, 'The Beatles' }
--[[
---
- - [4, 'The Beatles', 1960]
...
--]]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Select tuples by the specified partial key value --
select_multipart_partial = bands.index.year_band:select { 1965 }
--[[
---
- - [5, 'Pink Floyd', 1965]
  - [2, 'Scorpions', 1965]
  - [7, 'The Doors', 1965]
...
--]]

-- Select maximum 3 tuples by the specified secondary index --
select_limit = bands.index.band:select({}, { limit = 3 })
--[[
---
- - [3, 'Ace of Base', 1987]
  - [9, 'Led Zeppelin', 1968]
  - [8, 'Nirvana', 1987]
...
--]]

-- Select maximum 3 tuples with the key value greater than 1965 --
select_greater = bands.index.year:select({ 1965 }, { iterator = 'GT', limit = 3␣
↪})
--[[
---
- - [9, 'Led Zeppelin', 1968]
  - [10, 'Queen', 1970]
  - [1, 'Roxette', 1986]
...
--]]

-- Select maximum 3 tuples after the specified tuple --
select_after_tuple = bands.index.primary:select({}, { after = { 4, 'The Beatles
↪', 1960 }, limit = 3 })
--[[
---
- - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
  - [7, 'The Doors', 1965]
...
--]]

-- Select first 3 tuples and fetch a last tuple's position --
result, position = bands.index.primary:select({}, { limit = 3, fetch_pos = true␣
↪})
-- Then, pass this position as the 'after' parameter --
select_after_position = bands.index.primary:select({}, { limit = 3, after =␣
↪position })
--[[
---
- - [4, 'The Beatles', 1960]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
...
--]]
```

**Примечание:** `box.space.`*имя-спейса*`.index.`*имя-индекса*`:select(...)[1]` можно заменить `box.space.`*имя-спейса*`.index.`*имя-индекса*`:get(...)`. А именно, `get` можно использовать в качестве удобного сокращения для получения первого кортежа в наборе кортежей, который был бы выведен по запросу `select`. Однако, если в наборе кортежей больше одного кортежа, `get` завершится с ошибкой.

### index_object:get()

```
object index_object
```

index_object:get(*key*)

Search for a tuple via the given index, as described in the *select* topic.

**Параметры**

- `index_object` (*index_object*) – *ссылка на объект*.
- `key` (`scalar/table`) – значения для сопоставления с ключом индекса

**возвращает** кортеж, в котором поля ключа в индексе равны переданным значениям ключа.

**тип возвращаемого значения** кортеж

**Возможные ошибки:**

- отсутствие такого индекса;
- неправильный тип;
- больше одного кортежа подходят.

**Факторы сложности:** Размер индекса, тип индекса. См. также *space_object:get()*.

**Пример:**

```
tarantool> box.space.tester.index.primary:get(2)
---
- [2, 'Music']
...
```

### index_object:min()

```
object index_object
```

index_object:min([*key*])

Поиск минимального значения в указанном индексе.

**Параметры**

- index_object (*index_object*) – *ссылка на объект*.
- key (scalar/table) – значения для сопоставления с ключом индекса

**возвращает** кортеж для первого ключа в индексе. Если указано необязательное значение ключа key, будет выведен первый ключ, который больше или равен значению ключа key. Начиная с версии *Tarantool 2.0.4*, index:min(key) не вернет ничего, если индекс не содержит значения key.

**тип возвращаемого значения** кортеж

**Возможные ошибки:**

- Тип индекса не „TREE“.
- ER_TRANSACTION_CONFLICT, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Факторы сложности:** Размер индекса, тип индекса.

**Пример:**

Below are few examples of using min. To try out these examples, you need to bootstrap a Tarantool database as described in *Using data operations*.

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.space.bands:insert { 5, 'Pink Floyd', 1965 }
box.space.bands:insert { 6, 'The Rolling Stones', 1962 }
box.space.bands:insert { 7, 'The Doors', 1965 }
box.space.bands:insert { 8, 'Nirvana', 1987 }
box.space.bands:insert { 9, 'Led Zeppelin', 1968 }
box.space.bands:insert { 10, 'Queen', 1970 }

-- Find the minimum value in the specified index
min = box.space.bands.index.year:min()
--[[
---
- [4, 'The Beatles', 1960]
...
--]]

-- Find the minimum value that matches the partial key value
min_partial = box.space.bands.index.year_band:min(1965)
--[[
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- [5, 'Pink Floyd', 1965]
...
--]]
```

### index_object:max()

```
object index_object
```

index_object:max([*key*])

Поиск максимального значения в указанном индексе.

**Параметры**

- index_object (*index_object*) – *ссылка на объект*
- key (scalar/table) – значения для сопоставления с ключом индекса

**возвращает** кортеж для последнего ключа в индексе. Если указано необязательное значение ключа key, будет выведен последний ключ, который меньше или равен значению ключа key. Начиная с версии *Tarantool 2.0.4*, index:max(key) не возвращает ничего, если индекс не содержит значения key.

**тип возвращаемого значения** tuple

**Возможные ошибки:**

- Тип индекса не „TREE“.
- ER_TRANSACTION_CONFLICT, если транзакция стала конфликтной в *транзакционном режиме MVCC*.

**Факторы сложности:** размер индекса, тип индекса.

**Пример:**

Below are few examples of using max. To try out these examples, you need to bootstrap a Tarantool database as described in *Using data operations*.

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.space.bands:insert { 5, 'Pink Floyd', 1965 }
box.space.bands:insert { 6, 'The Rolling Stones', 1962 }
box.space.bands:insert { 7, 'The Doors', 1965 }
box.space.bands:insert { 8, 'Nirvana', 1987 }
box.space.bands:insert { 9, 'Led Zeppelin', 1968 }
box.space.bands:insert { 10, 'Queen', 1970 }

-- Find the maximum value in the specified index
max = box.space.bands.index.year:max()
--[[
---
- [8, 'Nirvana', 1987]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
--]]

-- Find the maximum value that matches the partial key value
max_partial = box.space.bands.index.year_band:max(1965)
--[[
---
- [7, 'The Doors', 1965]
...
--]]
```

### index_object:random()

```
object index_object
```

index_object:random(*seed*)

Поиск случайного значения в заданном индексе. Данный метод используется, когда важно получить представление о распределении данных в индексе без необходимости проходить по всему набору данных.

**Параметры**

- index_object (*index_object*) – *ссылка на объект*.
- seed (number) – произвольное неотрицательное целое число

**возвращает** кортеж для случайного ключа в индексе.

**тип возвращаемого значения** tuple

**Факторы сложности:** Размер индекса, тип индекса

**Note regarding storage engine:** vinyl does not support random().

**Пример:**

```
tarantool> box.space.tester.index.secondary:random(1)
---
- ['Beta!', 66, 'This is the second tuple!']
...
```

### index_object:count()

```
object index_object
```

index_object:count(*key*, *iterator*)

Итерация по индексу с подсчетом количества кортежей, которые соответствуют паре ключ-значение.

**Параметры**

- index_object (*index_object*) – *ссылка на объект*.
- key (scalar/table) – значения для сопоставления с ключом индекса

- `iterator` – метод сопоставления

**возвращает** количество совпадающих кортежей.

**тип возвращаемого значения** число

**Пример:**

Below are few examples of using `count`. To try out these examples, you need to bootstrap a Tarantool database as described in *Using data operations*.

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.space.bands:insert { 5, 'Pink Floyd', 1965 }
box.space.bands:insert { 6, 'The Rolling Stones', 1962 }
box.space.bands:insert { 7, 'The Doors', 1965 }
box.space.bands:insert { 8, 'Nirvana', 1987 }
box.space.bands:insert { 9, 'Led Zeppelin', 1968 }
box.space.bands:insert { 10, 'Queen', 1970 }

-- Count the number of tuples that match the full key value
count = box.space.bands.index.year:count(1965)
--[[
---
- 3
...
--]]

-- Count the number of tuples that match the partial key value
count_partial = box.space.bands.index.year_band:count(1965)
--[[
---
- 3
...
--]]
```

### index_object:update()

`object index_object`

`index_object:update(`*key*, *{{operator, field_identifier, value}, ...}*`)`
Обновление кортежа.

То же, что и *box.space...update()*, но поиск ключа происходит в этом индексе, вместо первичного. Данный индекс должен быть уникальным.

**Параметры**

- `index_object` (*index_object*) – *ссылка на объект*.
- `key` (scalar/table) – значения для сопоставления с ключом индекса
- `operator` (*string*) – тип операции, представленный строкой

- `field_identifier` (`field-or-string`) – к какому полю применяется операция. Номер поля может быть отрицательным, что означает, что позиция рассчитывается с конца кортежа. (#кортеж + отрицательный номер поля + 1)
- `value` (`lua_value`) – какое значение применяется

**возвращает**

- обновленный кортеж
- nil, если ключ не найден

**тип возвращаемого значения** tuple или nil

Начиная с версии Tarantool 2.3 кортеж можно обновить с помощью *JSON-путей*.

### index_object:delete()

`object index_object`

`index_object:delete(`*key*`)`

Удаление кортежа по ключу.

То же, что и *box.space... delete()*, но поиск ключа происходит в этом индексе, вместо первичного. Данный индекс должен быть уникальным.

**Параметры**

- `index_object` (*index_object*) – *ссылка на объект*.
- `key` (`scalar/table`) – значения для сопоставления с ключом индекса

**возвращает** удаленный кортеж.

**тип возвращаемого значения** кортеж

**Note regarding storage engine:** vinyl will return `nil`, rather than the deleted tuple.

### index_object:alter()

`object index_object`

`index_object:alter(`*{options}*`)`

Изменение индекса. В определенных обстоятельствах можно изменять некоторые характеристики индекса, например тип, параметры последовательности и определение его уникальности. Тем не менее, это обычно приводит к перестроению спейса за исключением простого случая, когда значение флага `is_nullable` меняется с `false` на `true`.

**Параметры**

- `index_object` (*index_object*) – *ссылка на объект*.
- `options` (*table*) – index options (see *index_opts*)

**возвращает** nil

**Возможные ошибки:**

- index does not exist
- the primary-key index cannot be changed to `{unique = false}`

**Примечание:** Vinyl does not support `alter()` of a primary-key index unless the space is empty.

**Пример 1:**

Можно добавлять и удалять поля, которые составляют первичный индекс:

```
tarantool> s = box.schema.create_space('test')
---
...
tarantool> i = s:create_index('i', {parts = {{field = 1, type = 'unsigned'}}})
---
...
tarantool> s:insert({1, 2})
---
- [1, 2]
...
tarantool> i:select()
---
- - [1, 2]
...
tarantool> i:alter({parts = {{field = 1, type = 'unsigned'}, {field = 2, type =
↪'unsigned'}}})
---
...
tarantool> s:insert({1, 't'})
---
- error: 'Tuple field 2 type does not match one required by operation: expected␣
↪unsigned'
...
```

**Пример 2:**

Можно изменять опции индекса для спейсов как в memtx'е, так и в vinyl'е:

```
tarantool> box.space.space55.index.primary:alter({type = 'HASH'})
---
...

tarantool> box.space.vinyl_space.index.i:alter({page_size=4096})
---
...
```

### index_object:drop()

`object index_object`

`index_object:drop()`

Удаление индекса. Побочный эффект удаления первичного индекса – все кортежи удалятся.

**Параметры**

- `index_object` (*index_object*) – *ссылка на объект*.

**возвращает** nil.

**Возможные ошибки:**

- индекс не существует,
- первичный индекс невозможно удалить, если существует вторичный индекс.

**Пример:**

```
tarantool> box.space.space55.index.primary:drop()
---
...
```

### index_object:rename()

```
object index_object
```

index_object:rename(*index-name*)

Переименование индекса.

**Параметры**

- index_object (*index_object*) – *ссылка на объект*.
- index-name (*string*) – новое имя индекса

**возвращает** nil

**Возможные ошибки:** index_object не существует.

**Пример:**

```
tarantool> box.space.space55.index.primary:rename('secondary')
---
...
```

**Факторы сложности:** Размер индекса, тип индекса, количество кортежей, к которым получен доступ.

### index_object:bsize()

```
object index_object
```

index_object:bsize()

Выдача общего количества байтов, занятых индексом.

**Параметры**

- index_object (*index_object*) – *ссылка на объект*.

**возвращает** количество байтов

**тип возвращаемого значения** number

### index_object:stat()

```
object index_object
```

index_object:stat()

Получение статистики о предпринятых действиях, которые влияют на индекс.

Используется с движком базы данных vinyl.

Подробные данные в выводе `index_object:stat()`:

- `index_object:stat().latency` содержит отметки времени в процентах;
- `index_object:stat().bytes` содержит общее количество байтов;
- `index_object:stat().disk.rows` содержит примерное количество кортежей в каждом диапазоне;
- `index_object:stat().disk.statement` содержит количество вставок, обновлений, обновлений и вставок, удалений (inserts|updates|upserts|deletes);
- `index_object:stat().disk.compaction` содержит количество слияний и их объем;
- `index_object:stat().disk.dump` содержит количество дампов и их объем;
- `index_object:stat().disk.iterator.bloom` содержит количество совпадений и несовпадений по фильтрами Блума;
- `index_object:stat().disk.pages` содержит размер в страницах;
- `index_object:stat().disk.last_level` содержит объем данных на последнем уровне LSM-дерева;
- `index_object:stat().cache.evict` содержит количество освобождений кэша;
- `index_object:stat().range_size` содержит максимальное количество байтов в диапазоне;
- `index_object:stat().dumps_per_compaction` содержит среднее число дампов, которое необходимо для запуска значительного слияния в любом диапазоне LSM-дерева.

Summary index statistics are also available via *box.stat.vinyl()*.

**Параметры**

- `index_object` (*index_object*) – *ссылка на объект*.

**возвращает** статистические данные

**тип возвращаемого значения** таблица

### index_object:compact()

```
object index_object
```

index_object:compact()

Удаление неиспользуемого пространства индекса. Для движка базы данных memtx метод бесполезен; `index_object:compact()` используется только для движка vinyl. Например, на движке vinyl при удалении кортежа память не возвращается незамедлительно. Существует планировщик автоматического восстановления ресурсов на основании таких факторов, как

форма LSM-дерева и усложнение, как описано в разделе *Хранение данных с помощью vinyl*, поэтому выполнять `index_object:compact()` вручную необходимости нет.

**возвращает** nil (Tarantool возвращает нулевое значение сразу же, не ожидая завершения слияния)

### index_object:tuple_pos()

`object index_object`

`index_object:tuple_pos(`*tuple*`)`

Return a tuple's position for an index. This value can be passed to the `after` option of the `select` and `pairs` methods:

- *index_object:select* and *space_object:select*
- *index_object:pairs* and *space_object:pairs*

Note that `tuple_pos` does not work with *functional* and multikey indexes.

**Параметры**

- `index_object` (*index_object*) – an *object reference*
- `tuple` (`scalar/table`) – a tuple whose position should be found

**Return** a tuple's position in a space

**Rtype** base64-encoded string

**Example:**

To try out this example, you need to bootstrap a Tarantool instance as described in *Using data operations*.

```
-- Insert test data --
tarantool> bands:insert{1, 'Roxette', 1986}
           bands:insert{2, 'Scorpions', 1965}
           bands:insert{3, 'Ace of Base', 1987}
           bands:insert{4, 'The Beatles', 1960}
           bands:insert{5, 'Pink Floyd', 1965}
           bands:insert{6, 'The Rolling Stones', 1962}
---
...

-- Get a tuple's position --
tarantool> position = bands.index.primary:tuple_pos({3, 'Ace of Base', 1987})
---
...
-- Pass the tuple's position as the 'after' parameter --
tarantool> bands:select({}, {limit = 3, after = position})
---
- - [4, 'The Beatles', 1960]
  - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
...
```

### index_object extensions

Пользователи могут сами определять любые желаемые функции и связывать их с индексами: фактически они могут создавать собственные методы для работы с индексом. Это можно сделать так:

1. создать Lua-функцию,
2. добавить имя функции в заданную глобальную переменную с типом «таблица» (table),
3. впоследствии в любое время, пока работает сервер, вызвать функцию с помощью `объект_индекса:имя-функции([параметры])`.

Есть три заданные глобальные переменные:

- Adding to `box_schema.index_mt` makes the function available for all indexes.
- Adding to `box_schema.memtx_index_mt` makes the function available for all memtx indexes.
- Adding to `box_schema.vinyl_index_mt` makes the function available for all vinyl indexes.

Можно также сделать задаваемый пользователем метод доступным только для одного индекса путем вызова `getmetatable(объект_индекса)` и последующего добавления имени функции в метатаблицу.

**Example 1:**

The example below shows how to extend all memtx indexes with the custom function:

```
box.schema.space.create('tester1', { engine = 'memtx' })
box.space.tester1:create_index('index1')
global_counter = 5

-- Create a custom function.
function increase_global_counter()
    global_counter = global_counter + 1
end

-- Extend all memtx indexes with the created function.
box.schema.memtx_index_mt.increase_global_counter = increase_global_counter

-- Call the 'increase_global_counter' function on 'index1'
-- to change the 'global_counter' value from 5 to 6.
box.space.tester1.index.index1:increase_global_counter()
```

**Example 2:**

The example below shows how to extend the specified index with the custom function with parameters:

```
box.schema.space.create('tester2', { engine = 'memtx', id = 1000 })
box.space.tester2:create_index('index2')
local_counter = 0

-- Create a custom function.
function increase_local_counter(i_arg, param)
    local_counter = local_counter + param + i_arg.space_id
end

-- Extend only the 'index2' index with the created function.
box.schema.memtx_index_mt.increase_local_counter = increase_local_counter
meta = getmetatable(box.space.tester2.index.index2)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
meta.increase_local_counter = increase_local_counter

-- Call the 'increase_local_counter' function on 'index2'
-- to change the 'local_counter' value from 0 to 1005.
box.space.tester2.index.index2:increase_local_counter(5)
```

### Вложенный модуль box.info

Вложенный модуль `box.info` предоставляет доступ к информации о переменных экземпляра сервера.

- **cluster.uuid** – это уникальный идентификатор набора реплик (UUID). У каждого экземпляра в наборе реплик будет одно и то же значение `cluster.uuid`. Данное значение также хранится в системном спейсе *box.space._schema*.
- **gc()** возвращает состояние *сборщика мусора в Tarantool*, в том числе контрольные точки и их потребителей (пользователи); более подробную информацию см. *здесь*.
- **id** corresponds to `replication[n].id` (see *here*).
- **lsn** corresponds to `replication[n].lsn` (see *here*).
- **listen** returns a real address to which an instance was bound (see *here*).
- **memory()** возвращает статистику об использовании памяти (см. *здесь*).
- **pid** – идентификатор процесса. Это значение также отображается с помощью модуля *tarantool* и по команде `ps -A` в Linux.
- **ro** is `true` if the instance is in read-only mode (same as *read_only* in `box.cfg{}`), or if status is „orphan“.
- **ro_reason** is `nil` if the instance is in writable mode. When the field is not `nil`, it contains the reason why the instance is read-only. Possible error reasons: `election`, `synchro`, `config`, and `orphan` (see *box.info.ro_reason* for details).
- **signature** представляет собой сумму всех значений `lsn` из векторных часов (**vclock**) всех экземпляров в наборе реплик.
- **sql().cache.size** – это количество байтов в кэше подготовленных операторов SQL.
- **sql().cache.stmt_count** – это количество операторов в кэше подготовленных операторов SQL.
- **status** – это текущий статус экземпляра. Он может быть:
  - `running` – экземпляр запущен,
  - `loading` – экземпляр восстанавливается из xlog'ов или snapshot'ов или стартует с нуля (bootstrapping),
  - `orphan` – экземпляр (еще) не подключился к необходимому количеству мастеров (см. *статус orphan*),
  - `hot_standby` – экземпляр является *резервным* для другого экземпляра.
- **uptime** – это количество секунд с момента запуска экземпляра. Данное значение также можно получить с помощью *tarantool.uptime()*.
- **uuid** corresponds to `replication[n].uuid` (see *here*).
- **vclock** – это таблица со значениями vclock всех экземпляров набора реплик, в которых произошли изменения данных.

- **version** – это версия Tarantool. Данное значение также можно отобразить с помощью команды *tarantool -V*.
- **vinyl** возвращает статистику времени работы для движка базы данных vinyl. Данная функция объявлена устаревшей, используйте *box.stat.vinyl()*.
- **election** shows the current state of a replica set node regarding leader election (see *here*).

Below is a list of all `box.info` functions and members.

| Имя | Назначение |
|---|---|
| *box.info()* | Выдача всех ключей и значений, указанных в модуле |
| *box.info.gc()* | Выдача информации о сборщике мусора |
| *box.info.memory()* | Выдача информации об использовании памяти |
| *box.info.replication_anon()* | List all the anonymous replicas following the instance |
| *box.info.replication* | Выдача статистики по всем экземплярам в наборе реплик |
| *box.info.listen* | Return a real address to which an instance was bound |
| *box.info.election* | Show the current state of a replica set node in regards to leader election |
| *box.info.synchro* | Show the current state of synchronous replication |
| *box.info.ro_reason* | Show the current mode of an instance (writable or read-only) |
| *box.info.schema_version* | Show the database schema version |

### box.info()

`box.info()`

Поскольку содержимое вложенного модуля `box.info` является динамическим, невозможно провести итерацию по ключам с помощью Lua-функции `pairs()`. Для этой цели модуль `box.info()` создает и возвращает Lua-таблицу со всеми ключами и значениями во вложенном модуле.

**возвращает** ключи и значения во вложенном модуле

**тип возвращаемого значения** таблица

**Пример:**

Данный пример приводится для набора со схемой мастер-реплика, который включает в себя один мастер-экземпляр и один реплика-экземпляр. Запрос был отправлен с реплики-экземпляра.

```
tarantool> box.info()
---
- version: 2.4.0-251-gc44ed3c08
  id: 1
  ro: false
  uuid: 1738767b-afa3-4987-b485-c333cf83415b
  package: Tarantool
  cluster:
    uuid: 40ee7f0f-7070-4650-8883-801e7014407c
  listen: '[::1]:57122'
  replication:
    1:
      id: 1
      uuid: 1738767b-afa3-4987-b485-c333cf83415b
      lsn: 16
  signature: 16
  status: running
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  vinyl: []
  uptime: 21
  lsn: 16
  sql: []
  gc: []
  pid: 20293
  memory: []
  vclock: {1: 16}
...
```

### box.info.gc()

box.info.gc()

Функция **gc** в `box.info` дает пользователю `admin` полное представление о факторах, которые влияют на *сборщик мусора Tarantool*. Сборщик мусора сопоставляет значения vclock (*векторные часы*) пользователей и контрольных точек, поэтому взглянув на `box.info.gc()`, можно понять, почему сборщик мусора не удалил старые WAL-файлы или что он может вскоре удалить.

- **gc().consumers** – список пользователей, запросы которых могут затронуть сборку мусора.
- **gc().checkpoints** – список сохраненных контрольных точек.
- **gc().checkpoints[n].references** – список ссылок на контрольную точку.
- **gc().checkpoints[n].vclock** – значение vclock контрольной точки.
- **gc().checkpoints[n].signature** – сумма компонентов vclock контрольной точки.
- **gc().checkpoint_is_in_progress** – true если идет создание контрольной точки, в противном случае false.
- **gc().vclock** – vclock сборщика мусора.
- **gc().signature** – сумма компонентов контрольной точки сборщика мусора.

### box.info.memory()

box.info.memory()

Функция **memory** в `box.info` дает пользователю `admin` полное представление об экземпляре Tarantool.

---

**Примечание:** Чтобы получить представление о подсистеме vinyl'а, используйте *box.stat.vinyl()*.

---

- **memory().cache** – это количество байтов, используемых для кэширования данных пользователей. Движок базы данных memtx не нуждается в кэше, то есть на самом деле это количество байтов в кэше для кортежей движка базы данных vinyl.
- **memory().data** – количество байтов, используемых для хранения данных пользователей (кортежи) в движке memtx и на уровне 0 движка vinyl, не принимая во внимание фрагментацию памяти.
- **memory().index** – количество байтов, используемых для индексирования данных пользователей, включая экстенты для деревьев в memtx'е и vinyl'е, индекс страниц и фильтры Блума в vinyl'е.

- **memory().lua** – количество байтов, используемых Lua-интерпретатором.
- **memory().net** – количество байтов, используемых буферами для сетевого ввода-вывода.
- **memory().tx** – количество байтов, используемых активными транзакциями. Для движка базы данных vinyl это общий размер всех размещаемых объектов (структура `txv`, структура `vy_tx`, структура `vy_read_interval`) и кортежей, прикрепленных к этим объектам.

Пример с минимальным распределением, когда используется только движок базы данных memtx:

```
tarantool> box.info.memory()
---
- cache: 0
  data: 6552
  tx: 0
  lua: 1315567
  net: 98304
  index: 1196032
...
```

### box.info.replication_anon()

box.info.replication_anon()

Список всех *анонимных реплик* экземпляра.

Вывод аналогичен выводу `box.info.replication`, за исключением того, что анонимные реплики индексируются по их uuid, а не по идентификаторам серверов, так как идентификаторы серверов не имеют значения для анонимных реплик.

Обратите внимание, что при вызове `box.info.replication_anon`, возвращается только количество анонимных реплик текущего экземпляра. Чтобы увидеть полную статистику, нужно вызвать `box.info.replication_anon()`. Это делается для того, чтобы не перегружать вывод `box.info` избыточной информацией, так как анонимных реплик может быть много.

**Пример:**

```
tarantool> box.info.replication_anon
---
- count: 2
...

tarantool> box.info.replication_anon()
---
- 3a6a2cfb-7e47-42f6-8309-7a25c37feea1:
    id: 0
    uuid: 3a6a2cfb-7e47-42f6-8309-7a25c37feea1
    lsn: 0
    downstream:
      status: follow
      idle: 0.76203499999974
      vclock: {1: 1}
  f58e4cb0-e0a8-42a1-b439-591dd36c8e5e:
    id: 0
    uuid: f58e4cb0-e0a8-42a1-b439-591dd36c8e5e
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    lsn: 0
    downstream:
      status: follow
      idle: 0.0041349999992235
      vclock: {1: 1}
...
```

Нужно отметить, что анонимные реплики скрывают свои `lsn` от других, поэтому `lsn` анонимной реплики всегда будет записан как ноль, даже если анонимная реплика выполняет некоторые локальные операции на спейсе. Для того, чтобы узнать `lsn` конкретной анонимной реплики, необходимо вызвать `box.info.lsn` на ней.

### box.info.replication

`box.info.replication`

The **replication** section of `box.info()` is a table with statistics for all instances in the replica set that the current instance belongs to. To see the example, refer to *Monitoring a replica set*.

In the following, *n* is the index number of one table item, for example, `replication[1]`, which has data about server instance number 1, which may or may not be the same as the current instance (the «current instance» is what is responding to `box.info`).

- `replication[n].id` – это короткий числовой идентификатор экземпляра в наборе реплик. Данное значение хранится в системном спейсе *box.space._cluster*.
- `replication[n].uuid` – это глобально-уникальный идентификатор экземпляра. Данное значение хранится в системном спейсе *box.space._cluster*.
- `replication[n].lsn` is the *log sequence number* (LSN) for the latest entry in instance *n*'s *write-ahead log* (WAL).
- `replication[n].upstream` appears (is not `nil`) if the current instance is following or intending to follow instance *n*, which ordinarily means `replication[n].upstream.status` = `follow`, `replication[n].upstream.peer` = url of instance *n* which is being followed, `replication[n].lag and idle` = the instance's speed, described later. Another way to say this is: `replication[n].upstream` will appear when `replication[n].upstream.peer` is not of the current instance, and is not read-only, and was specified in `box.cfg{replication={...}}`, so it is shown in *box.cfg.replication*.
- `replication[n].upstream.status` is the replication status of the connection with the instance *n*:
  - `connect`: an instance is connecting to the master.
  - `auth`: *authentication* is being performed.
  - `wait_snapshot`: an instance is receiving metadata from the master. If join fails with a non-critical *error* at this stage (for example, `ER_READONLY`, `ER_ACCESS_DENIED`, or a network-related issue), an instance tries to find a new master to join.
  - `fetch_snapshot`: an instance is receiving data from the master's `.snap` files.
  - `final_join`: an instance is receiving new data added during `fetch_snapshot`.
  - `sync`: the master and replica are synchronizing to have the same data.

  - `follow`: the current instance's role is **replica**. This means that the instance is read-only or acts as a replica for this remote peer in *master-master* configuration. The instance is receiving or able to receive data from the instance *n*'s (upstream) master.
  - `stopped`: replication is stopped due to a replication error (for example, *duplicate key*).
  - `disconnected`: an instance is not connected to the replica set (for example, due to network issues, not replication errors).

  Learn more from *Replication stages*.

- `replication[n].upstream.idle` is the time (in seconds) since the last event was received. This is the primary indicator of replication health. Learn more from *Monitoring a replica set*.

- `replication[n].upstream.peer` contains instance *n*'s *URI*, for example, 127.0.0.1:3302. Learn more from *Monitoring a replica set*.

- `replication[n].upstream.lag` is the time difference between the local time of instance *n*, recorded when the event was received, and the local time at another master recorded when the event was written to the *write-ahead log* on that master. Learn more from *Monitoring a replica set*.

- `replication[n].upstream.message` contains an error message in case of a *degraded state*; otherwise, it is `nil`.

- `replication[n].downstream` appears (is not `nil`) with data about an instance that is following instance *n* or is intending to follow it, which ordinarily means `replication[n].downstream.status = follow`.

- `replication[n].downstream.vclock` contains the *vector clock*, which is a table of „**id**, **lsn**" pairs, for example, `vclock: {1: 3054773, 4: 8938827, 3: 285902018}`. (Notice that the table may have multiple pairs although `vclock` is a singular name).

  Даже если экземпляр *удален*, его значения все равно появятся здесь; однако, его значения будут переопределены, если позже экземпляр присоединится с тем же UUID. Пары векторных часов будут появляться только если `lsn > 0`.

  `replication[n].downstream.vclock` может быть таким же, как и vclock текущего экземпляра (`box.info.vclock`), потому что все значения vclock в кластере известны. Мастер будет знать, что находится в копии vclock реплики, потому что, когда мастер делает изменение данных, он посылает информацию об изменении на реплику (включая векторные часы мастера), и реплика отвечает тем, что находится в ее таблице векторных часов.

  A replica also sends its entire vector clock table in response to a master's heartbeat message, see the heartbeat-message examples in the section *Binary protocol – replication*.

- `replication[n].downstream.idle` – это время (в секундах) с момента последней отправки событий экземпляром *n* через downstream-репликацию.

- `replication[n].downstream.status` – это статус для downstream-репликации:
  - `stopped` означает, что downstream-репликация остановлена,
  - `follow` означает, что downstream-репликация находится в процессе (экземпляр *n* готов принимать данные от мастера или уже делает это).

- `replication[n].downstream.lag` is the time difference between the local time at the master node, recorded when a particular transaction was written to the *write-ahead log*, and the local time recorded when it receives an acknowledgment for this transaction from a replica. Since version *2.10.0*. See more in *Monitoring a replica set*.

- `replication[n].downstream.message` and `replication[n].downstream.system_message` will be `nil` unless a problem occurs with the connection. For example, if instance *n* goes down, then one may see `status = 'stopped'`, `message = 'unexpected EOF when reading from socket'`, and `system_message = 'Broken pipe'`. See also *degraded state*.

### box.info.listen

`box.info.listen`

Since version *2.4.1*. Return a real address to which an instance was bound. For example, if `box.cfg{listen}` was set with a zero port, `box.info.listen` will show a real port. The address is stored as a string:

- unix/:<path> для доменных сокетов UNIX
- <ip>:<port> для IPv4
- [ip]:<port> для IPv6

Если экземпляр ничего не прослушивает, то `box.info.listen` вернет nil.

**Пример:**

```
tarantool> box.cfg{listen=0}
---
...
tarantool> box.cfg.listen
---
- '0'
...
tarantool> box.info.listen
---
- 0.0.0.0:44149
...
```

### box.info.election

`box.info.election`

Since version *2.6.1*.

Show the current state of a replica set node in regards to *leader election*. The following information is provided:

- `state` – the election state (mode) of the node. Possible values are `leader`, `follower`, or `candidate`. For more details, refer to description of the *leader election process*. When *replication.failover* is set to `election`, the node is writable only in the `leader` state.
- `term` – the current election term.
- `vote` – the ID of a node the current node votes for. If the value is `0`, it means the node hasn't voted in the current term yet.
- `leader` – a leader node ID in the current term. If the value is `0`, it means the node doesn't know which node is the leader in the current term.
- `leader_name` – a leader name. Returns `nil` if there is no leader in a cluster or *box.NULL* if a leader does not have a name. Since version *3.0.0*.

- `leader_idle` – time in seconds since the last interaction with the known leader. Since version *2.10.0*.

---

**Примечание:** IDs in the `box.info.election` output are the replica IDs visible in the `box.info.id` output on each node and in the `_cluster` space.

---

**Пример:**

```
auto_leader:instance001> box.info.election
---
- leader_idle: 0
  leader_name: instance001
  state: leader
  vote: 2
  term: 3
  leader: 2
...
```

See also: *Master-replica: automated failover*.

### box.info.synchro

box.info.synchro
Since version *2.8.1*.

Show the current state of synchronous replication.

In *synchronous replication*, transaction is considered committed only after achieving the required quorum number. While transactions are collecting confirmations from remote nodes, these transactions are waiting in the queue.

The following information is provided:

- queue:
  - `owner` (since version *2.10.0*) – ID of the replica that owns the synchronous transaction queue. Once an owner instance appears, all other instances become read-only. If the `owner` field is `0`, then every instance may be writable, but they can't create any synchronous transactions. To claim or reclaim the queue, use *box.ctl.promote()* on the instance that you want to promote. With elections enabled, an instance runs `box.ctl.promote()` command automatically after winning the elections. To clear the ownership, call *box.ctl.demote()* on the synchronous queue owner.
  - `term` (since version *2.10.0*) – current queue term. It contains the term of the last `PROMOTE` request. Usually, it is equal to *box.info.election.term*. However, the queue term value may be less than the election term. It can happen when a new round of elections has started, but no instance has been promoted yet.
  - `len` – the number of entries that are currently waiting in the queue.
  - `busy` (since version *2.10.0*) – the boolean value is `true` when the instance is processing or writing some system request that modifies the queue (for example, `PROMOTE`, `CONFIRM`, or `ROLLBACK`). Until the request is complete, any other incoming synchronous transactions and system requests will be delayed.

- `quorum` – the resulting value of the *replication_synchro_quorum* configuration option. Since version *2.5.3*, the option can be set as a dynamic formula. In this case, the value of the `quorum` member depends on the current number of replicas.

**Example 1:**

In this example, the `quorum` field is equal to `1`. That is, synchronous transactions work like asynchronous ones. `1` means that a successful WAL writing to the master is enough to commit.

```
tarantool> box.info.synchro
---
- queue:
    owner: 1
    term: 2
    len: 0
    busy: false
  quorum: 1
...
```

**Example 2:**

First, set a quorum number and a timeout for synchronous replication using the following command:

```
tarantool> box.cfg{
         > replication_synchro_quorum=2,
         > replication_synchro_timeout=1000
         > }
```

Next, check the current state of synchronous replication:

```
tarantool> box.info.synchro
---
- queue:
    owner: 1
    term: 2
    len: 0
    busy: false
  quorum: 2
...
```

Create a space called `sync` and enable synchronous replication on this space. Then, create an index.

```
tarantool> s = box.schema.space.create("sync", {is_sync=true})
tarantool> _ = s:create_index('pk')
```

After that, use `box.ctl.promote()` function to claim a queue:

```
tarantool> box.ctl.promote()
```

Next, perform data manipulations:

```
tarantool> require('fiber').new(function() box.space.sync:replace{1} end)
---
- status: suspended
  name: lua
  id: 119
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> require('fiber').new(function() box.space.sync:replace{1} end)
---
- status: suspended
  name: lua
  id: 120
...
tarantool> require('fiber').new(function() box.space.sync:replace{1} end)
---
- status: suspended
  name: lua
  id: 121
...
```

If you call the `box.info.synchro` command again, you will see that now there are 3 transactions waiting in the queue:

```
tarantool> box.info.synchro
---
- queue:
    owner: 1
    term: 2
    len: 3
    busy: false
  quorum: 2
...
```

### box.info.ro_reason

`box.info.ro_reason`

Since *2.10.0*. Show the current mode of an instance (writable or read-only). Contains `nil` if the instance is in writable mode. When the field is not `nil`, reports the reason why the instance is read-only.

Possible error reasons:

- `election` – the instance is not the leader. That is, `box.cfg.election_mode` is not `off`. See *box.info.election* for details.
- `synchro` – the instance is not the owner of the synchronous transaction queue. For details, see *box.info.synchro*.
- `config` – the server instance is in read-only mode. That is, *box.cfg.read_only* is `true`.
- `orphan` – the instance is in `orphan` state. For details, see *the orphan status page*.

**Rtype** string

**Example:**

```
tarantool> box.info.ro_reason
---
- null
...
```

### box.info.schema_version

box.info.schema_version
Since *2.11.0*. Show the database schema version. A schema version is a number that indicates whether the *database schema* is changed. For example, the `schema_version` value grows if a *space* or *index* is added or deleted, or a space, index, or field name is changed.

**Rtype** number

**Example:**

```
tarantool> box.info.schema_version
---
- 84
...
```

See also: *IPROTO_SCHEMA_VERSION*

### Submodule box.iproto

Since *2.11.0*.

The `box.iproto` submodule provides the ability to work with the network subsystem of Tarantool. It allows you to extend the *IPROTO* functionality from Lua. With this submodule, you can:

- *parse unknown IPROTO request types*
- *send arbitrary IPROTO packets*
- *override the behavior* of the existing and unknown request types in the binary protocol

The submodule exports all IPROTO *constants* and *features* to Lua.

### IPROTO constants

IPROTO constants in the `box.iproto` namespace are written in uppercase letters without the `IPROTO_` prefix. The constants are divided into several groups:

- *key*. Example: *IPROTO_SYNC*.
- *request type*. Example: *IPROTO_OK*.
- *flag*. Example: *IPROTO_COMMIT*.
- *ballot key*. Example: *IPROTO_FLAG_COMMIT*.
- *metadata key*. Example: *IPROTO_FIELD_IS_NULLABLE*.
- *RAFT key*. Example: *IPROTO_TERM*.

Each group is located in the corresponding subnamespace without the prefix. For example:

```
box.iproto.key.SYNC = 0x01
-- ...
box.iproto.type.SELECT = 1
-- ...
box.iproto.flag.COMMIT = 1
-- ...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
box.iproto.ballot_key.VCLOCK = 2
-- ...
box.iproto.metadata_key.IS_NULLABLE = 3
-- ...
box.iproto.raft_key.TERM = 0
-- ...
```

### IPROTO features

The submodule exports:

- the current IPROTO protocol version (*box.iproto.protocol_version*)
- the set of IPROTO protocol features supported by the server (*box.iproto.protocol_features*)
- IPROTO protocol features with the corresponding code (*box.iproto.feature*)

**Example**

The example converts the feature names from `box.iproto.protocol_features` set into codes:

```
-- Features supported by the server
box.iproto.protocol_features = {
    streams = true,
    transactions = true,
    error_extension = true,
    watchers = true,
    pagination = true,
}

-- Convert the feature names into codes
features = {}
for name in pairs(box.iproto.protocol_features) do
    table.insert(features, box.iproto.feature[name])
end
return features -- [0, 1, 2, 3, 4]
```

### Handling the unknown IPROTO request types

Every IPROTO request has a static handler. That is, before version *2.11.0*, any unknown request raised an error. Since *2.11.0*, a new request type is introduced – *IPROTO_UNKNOWN*. This type is used to override the handlers of the unknown IPROTO request types. For details, see *box.iproto.override()* and *box_iproto_override* functions.

### API reference

The table lists all available functions and data of the submodule:

| Name | Use |
|---|---|
| *box.iproto.key* | Request keys |
| *box.iproto.type* | Request types |
| *box.iproto.flag* | Flags from the *IPROTO_FLAGS* key |
| *box.iproto.ballot_key* | Keys from the *IPROTO_BALLOT* requests |
| *box.iproto.metadata_key* | Keys nested in the *IPROTO_METADATA* key |
| *box.iproto.raft* | Keys from the `IPROTO_RAFT_` requests |
| *box.iproto.protocol_version* | The current IPROTO protocol version |
| *box.iproto.protocol_features* | The set of supported IPROTO protocol features |
| *box.iproto.feature* | IPROTO protocol *features* |
| *box.iproto.override()* | Set a new IPROTO request handler callback for the given request type |
| *box.iproto.send()* | Send an IPROTO packet over the session's socket |

### box.iproto.key

`box.iproto.key`

Contains all available request keys, except *raft*, *metadata*, and *ballot* keys. Learn more: *Keys used in requests and responses*.

**Example**

```
tarantool> box.iproto.key.SYNC
---
- 1
...
```

### box.iproto.type

`box.iproto.type`

Contains all available request types. Learn more about the requests: *Client-server requests and responses*.

**Example**

```
tarantool> box.iproto.type.UNKNOWN
---
- -1
...
tarantool> box.iproto.type.CHUNK
---
- 128
...
```

### box.iproto.flag

box.iproto.flag
: Contains the flags from the `IPROTO_FLAGS` key. Learn more: *IPROTO_FLAGS key*.

**Example**

```
tarantool> box.iproto.flag.COMMIT
---
- 1
...
tarantool> box.iproto.flag.WAIT_SYNC
---
- 2
...
```

### box.iproto.ballot_key

box.iproto.ballot_key
: Contains the keys from the *IPROTO_BALLOT* requests. Learn more: *IPROTO_BALLOT keys*.

**Example**

```
tarantool> box.iproto.ballot_key.IS_RO_CFG
---
- 1
...
tarantool> box.iproto.ballot_key.VCLOCK
---
- 2
...
```

### box.iproto.metadata_key

box.iproto.metadata_key
: Contains the `IPROTO_FIELD_*` keys, which are nested in the *IPROTO_METADATA* key.

**Example**

```
tarantool> box.iproto.metadata_key.NAME
---
- 0
...
tarantool> box.iproto.metadata_key.TYPE
---
- 1
...
```

### box.iproto.raft

box.iproto.raft_key
: Contains the keys from the `IPROTO_RAFT_*` requests. Learn more: *Synchronous replication keys*.

**Example**

```
tarantool> box.iproto.raft_key.TERM
---
- 0
...
tarantool> box.iproto.raft_key.VOTE
---
- 1
...
```

### box.iproto.protocol_version

box.iproto.protocol_version
: The current IPROTO protocol version of the server. Learn more: *IPROTO_ID*.

**Example**

```
tarantool> box.iproto.protocol_version
---
- 4
...
```

### box.iproto.protocol_features

box.iproto.protocol_features
: The set of IPROTO protocol features supported by the server. Learn more: *net.box features*, src/box/iproto_features.h, and iproto_features_resolve().

**Example**

```
tarantool> box.iproto.protocol_features
---
- transactions: true
  watchers: true
  error_extension: true
  streams: true
  pagination: true
...
```

### box.iproto.feature

box.iproto.feature
Contains the IPROTO protocol features that are supported by the server. Each feature is mapped to its corresponding code. Learn more: *IPROTO_FEATURES*.

The features in the namespace are written

- in lowercase letters
- without the IPROTO_FEATURE_ prefix

**Example**

```
tarantool> box.iproto.feature.streams
---
- 0
...
tarantool> box.iproto.feature.transactions
---
- 1
...
```

### box.iproto.override()

box.iproto.override(*request_type*, *handler*)
Since version *2.11.0*. Set a new IPROTO request handler callback for the given request type.

**Параметры**

- request_type (number) – a request type code. Possible values:
  - a type code from *box.iproto.type* (except box.iproto.type.UNKNOWN) – override the existing request type handler.
  - box.iproto.type.UNKNOWN – override the handler of unknown request types.
- handler (function) – IPROTO request handler. The signature of a handler function: function(sid, header, body), where
  - header (userdata): a request header encoded as a *msgpack_object*
  - body (userdata): a request body encoded as a *msgpack_object*

  Returns true on success, otherwise false. On false, there is a fallback to the default handler. Also, you can indicate an error by throwing an exception. In this case, the return value is false, but this does not always mean a failure.

  To reset the request handler, set the handler parameter to nil.

**Return** none

**Possible errors:**

If a Lua handler throws an exception, the behavior is similar to that of a remote procedure call. The following errors are returned to the client over IPROTO (see src/lua/utils.h):

- ER_PROC_LUA – an exception is thrown from a Lua handler, diagnostic is not set.
- diagnostics from src/box/errcode.h – an exception is thrown, diagnostic is set.

For details, see src/box/errcode.h.

**Предупреждение:** When using `box.iproto.override()`, it is important that you follow the wire protocol. That is, the server response should match the return value types of the corresponding request type. Otherwise, it could lead to peer breakdown or undefined behavior.

**Example:**

Define a handler function for the `box.iproto.type.SELECT` request type:

```
local function iproto_select_handler_lua(header, body)
    if body.space_id == 512 then
        box.iproto.send(box.session.id(),
                { request_type = box.iproto.type.OK,
                  sync = header.SYNC,
                  schema_version = box.info.schema_version },
                { data = { 1, 2, 3 } })
        return true
    end
    return false
end
```

Override `box.iproto.type.SELECT` handler:

```
box.iproto.override(box.iproto.type.SELECT, iproto_select_handler_lua)
```

Reset `box.iproto.type.SELECT` handler:

```
box.iproto.override(box.iproto.type.SELECT, nil)
```

Override a handler function for the unknown request type:

```
box.iproto.override(box.iproto.type.UNKNOWN, iproto_unknown_request_handler_lua)
```

### box.iproto.send()

`box.iproto.send`(*sid*, *header*[, *body*])

Since version *2.11.0*. Send an *IPROTO* packet over the session's socket with the given MsgPack header and body. The header and body contain exported IPROTO constants from the *box.iproto()* submodule. Possible IPROTO constant formats:

- a lowercase constant without the `IPROTO_` prefix (`schema_version`, `request_type`)
- a constant from the corresponding *box.iproto* subnamespace (`box.iproto.SCHEMA_VERSION`, `box.iproto.REQUEST_TYPE`)

The function works for binary sessions only. For details, see *box.session.type()*.

**Параметры**

- `sid` (`number`) – the IPROTO session identifier (see *box.session.id()*)
- `header` (`table|string`) – a request header encoded as MsgPack
- `body` (`table|string|nil`) – a request body encoded as MsgPack

**Return** 0 on success, otherwise an error is raised

**Rtype** number

**Possible errors:**

- `ER_SESSION_CLOSED` – the session is closed.
- `ER_NO_SUCH_SESSION` – the session does not exist.
- `ER_MEMORY_ISSUE` – out-of-memory limit has been reached.
- `ER_WRONG_SESSION_TYPE` – the session type is not binary.

For details, see src/box/errcode.h.

**Examples:**

Send a packet using Lua tables and string IPROTO constants as keys:

```
box.iproto.send(box.session.id(),
        { request_type = box.iproto.type.OK,
          sync = 10,
          schema_version = box.info.schema_version },
        { data = 1 })
```

Send a packet using Lua tables and numeric IPROTO constants:

```
box.iproto.send(box.session.id(),
        { [box.iproto.key.REQUEST_TYPE] = box.iproto.type.OK,
          [box.iproto.key.SYNC] = 10,
          [box.iproto.key.SCHEMA_VERSION] = box.info.schema_version },
        { [box.iproto.key.DATA] = 1 })
```

Send a packet that contains only the header:

```
box.iproto.send(box.session.id(),
        { request_type = box.iproto.type.OK,
          sync = 10,
          schema_version = box.info.schema_version })
```

### Submodule box.read_view

The box.read_view submodule contains functions related to read views.

| Name | Use |
|---|---|
| *box.read_view.list()* | Return an array of all active database read views. |

### box.read_view.list()

`read_view.list()`

Return an array of all active database read views. This array might include the following read view types:

- *read views* created by application code (Enterprise Edition only)
- system read views (used, for example, to make a *checkpoint* or join a new *replica*)

Read views created by application code also have the `space` field. The field lists all spaces available in a read view, and may be used like a read view object returned by `box.read_view.open()`.

---

**Примечание:** `read_view.list()` also contains read views created using the *C API* (*box_raw_read_view_new()*). Note that you cannot access database spaces included in such views from Lua.

---

**Example:**

```
tarantool> box.read_view.list()
---
- - timestamp: 1138.98706933
    signature: 47
    is_system: false
    status: open
    vclock: &0 {1: 47}
    name: read_view1
    id: 1
  - timestamp: 1172.202995842
    signature: 49
    is_system: false
    status: open
    vclock: &1 {1: 49}
    name: read_view2
    id: 2
...
```

### Вложенный модуль box.schema

Вложенный модуль `box.schema` содержит функции для определения данных для спейсов, пользователей, ролей, кортежей и последовательностей.

Ниже приведен перечень всех функций модуля `box.schema`.

| Имя | Использование |
|---|---|
| *box.schema.space.create()* | Создание спейса |
| *box.schema.upgrade()* | Обновить базу данных |
| *box.schema.downgrade()* | Downgrade a database |
| *box.schema.downgrade_issues()* | List downgrade issues for the specified Tarantool version |
| *box.schema.downgrade_versions()* | List Tarantool versions available for downgrade |
| *box.schema.user.create()* | Создание пользователя |
| *box.schema.user.drop()* | Удаление пользователя |
| *box.schema.user.exists()* | Проверка существования пользователя |
| *box.schema.user.grant()* | Выдача прав пользователю или роли |
| *box.schema.user.revoke()* | Отмена прав пользователя или роли |
| *box.schema.user.password()* | Получение хеша пароля пользователя |
| *box.schema.user.passwd()* | Ассоциация пароля с пользователем |
| *box.schema.user.info()* | Получение описания прав пользователя |
| *box.schema.role.create()* | Создание роли |
| *box.schema.role.drop()* | Удаление роли |
| *box.schema.role.exists()* | Проверка наличия роли |
| *box.schema.role.grant()* | Выдача прав роли |
| *box.schema.role.revoke()* | Отмена прав роли |
| *box.schema.role.info()* | Получение описания прав роли |
| *box.schema.func.create()* | Создание кортежа с функцией |
| *box.schema.func.drop()* | Удаление кортежа с функцией |
| *box.schema.func.exists()* | Проверка наличия кортежа с функцией |
| *box.schema.func.reload()* | Перезагрузка модуля на C (со всеми его функциями) без перезапуска сервера |

### box.schema.upgrade()

`box.schema.upgrade()`

If you created a database with an older Tarantool version and have now installed a newer version, make the request `box.schema.upgrade()`. This updates Tarantool system spaces to match the currently installed version of Tarantool. You can learn about the general upgrade process from the *Upgrades* topic.

For example, here is what happens when you run `box.schema.upgrade()` with a database created with Tarantool version 1.6.4 to version 1.7.2 (only a small part of the output is shown):

```
tarantool> box.schema.upgrade()
alter index primary on _space set options to {"unique":true}, parts to [[0,"unsigned
↪"]]
alter space _schema set options to {}
create view _vindex...
grant read access to 'public' role for _vindex view
set schema version to 1.7.0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...
```

You can also put the request `box.schema.upgrade()` inside a *box.once()* function in your Tarantool *initialization file*. On startup, this will create new system spaces, update data type names (for example, `num` -> `unsigned`, `str` -> `string`) and options in Tarantool system spaces.

See also: *box.schema.downgrade()*

### box.schema.downgrade()

`box.schema.downgrade(`*version*`)`

Allows you to downgrade a database to the specified Tarantool version. This might be useful if you need to run a database on older Tarantool versions.

To prepare a database for using it on an older Tarantool instance, call `box.schema.downgrade` and pass the desired Tarantool version:

```
tarantool> box.schema.downgrade('2.8.4')
```

---

**Примечание:** The Tarantool's downgrade procedure is similar to the upgrade process that is described in the *Upgrades* topic. You need to run `box.schema.downgrade()` only on master and execute `box.shapshot()` on every instance in a replica set before restart to an older version.

---

To see Tarantool versions available for downgrade, call *box.schema.downgrade_versions()*. The oldest release available for downgrade is `2.8.2`.

Note that the downgrade process might fail if the database enables specific features not supported in the target Tarantool version. You can see all such issues using the *box.schema.downgrade_issues()* method, which accepts the target version. For example, `downgrade` to the `2.8.4` version fails if you use *tuple compression* or field *constraints* in your database:

```
tarantool> box.schema.downgrade_issues('2.8.4')
---
- - Tuple compression is found in space 'bands', field 'band_name'. It is supported
    starting from version 2.10.0.
  - Field constraint is found in space 'bands', field 'year'. It is supported␣
↪starting
    from version 2.10.0.
...
```

See also: *box.schema.upgrade()*

### box.schema.downgrade_versions()

box.schema.downgrade_versions()
Return a list of Tarantool versions available for downgrade. To learn how to downgrade a database to the specified Tarantool version, see *box.schema.downgrade()*.

**Return** a list of Tarantool versions

**Rtype** *table*

### box.schema.downgrade_issues()

box.schema.downgrade_issues(*version*)
Return a list of downgrade issues for the specified Tarantool version. To learn how to downgrade a database to the specified Tarantool version, see *box.schema.downgrade()*.

**Return** a list of downgrade issues

**Rtype** *table*

### box.schema.user.create()

box.schema.user.create(*name*[, *{options}*])
Создание пользователя. Чтобы получить информацию о том, как происходит управление данными пользователя в Tarantool, см. раздел *Пользователи* и справочник по спейсу *_user*.

Возможные параметры:

- `if_not_exists` (если отсутствует) = `true|false` (правда/ложь, по умолчанию ложь) - логическое значение boolean; `true` (правда) означает, что ошибка не выпадет, если пользователь уже существует,
- `password` (default = „“) - string; the `password` = *password* specification is good because in a *URI* (Uniform Resource Identifier) it is usually illegal to include a username without a password.

---

**Примечание:** Максимальное количество пользователей – 32.

---

**Параметры**

- `name` (*string*) – a user name, which should conform to the *rules for object names*
- `options` (*table*) – `if_not_exists`, `password`

**возвращает** nil

**Примеры:**

```
box.schema.user.create('testuser', { password = 'foobar' })
```

See also: *Managing users*.

### box.schema.user.drop()

box.schema.user.drop(*username*[, *{options}*])
Удаление пользователя. Чтобы получить информацию о том, как происходит управление данными пользователя в Tarantool, см. раздел *Пользователи* и справочник по спейсу *_user*.

**Параметры**

- **username** (*string*) – имя пользователя
- **options** (*table*) – **if_exists** (если существует) = **true**|**false** (правда/ложь, по умолчанию ложь) - логическое значение boolean; **true** (правда) означает, что ошибка не выпадет, если такой пользователь не существует.

**Примеры:**

```
box.schema.user.drop('testuser')
```

See also: *Managing users*.

### box.schema.user.exists()

box.schema.user.exists(*username*)
Выдача **true** (правда), если пользователь существует; выдача **false** (ложь), если пользователь отсутствует. Чтобы получить информацию о том, как происходит управление данными пользователя в Tarantool, см. раздел *Пользователи* и справочник по спейсу *_user*.

**Параметры**

- **username** (*string*) – имя пользователя

**тип возвращаемого значения** логическое значение bool

See also: *Getting a user's information*.

### box.schema.user.grant()

box.schema.user.grant(*username*, *permissions*, *object-type*, *object-name*[, *{options}*])
box.schema.user.grant(*username*, *permissions*, *'universe'*[, *nil*, *{options}*])
box.schema.user.grant(*username*, *role-name*[, *nil*, *nil*, *{options}*])
Выдача *прав* пользователю или другой роли.

**Параметры**

- **username** (*string*) – the name of a user to grant privileges to
- **permissions** (*string*) – one or more *permissions* to grant to the user (for example, **read** or **read,write**)
- **object-type** (*string*) – a database *object type* to grant permissions to (for example, **space**, **role**, or **function**)
- **object-name** (*string*) – the name of a database object to grant permissions to
- **role-name** (*string*) – the name of a role to grant to the user
- **options** (*table*) – **grantor**, **if_not_exists**

Если указана `'function','object-name'`, то должен существовать кортеж _func с этим именем объекта.

**Variation:** instead of `object-type, object-name` say `universe` which means „all object-types and all objects“. In this case, object name is omitted.

**Variation:** instead of `permissions, object-type, object-name` say `role-name` (see section *Roles*).

**Variation:** instead of `box.schema.user.grant('username','usage,session','universe',nil, {if_not_exists=true})` say `box.schema.user.enable('username')`.

Возможны следующие опции:

- `grantor` = *grantor_name_or_id* – строка или число, для пользователя, который выдает права,
- `if_not_exists` = `true|false` (default = `false`) - boolean; `true` означает, что ошибки не должно быть, если пользователь уже имеет права.

**Пример:**

```
box.schema.user.grant('testuser', 'read', 'space', 'writers')
box.schema.user.grant('testuser', 'read,write', 'space', 'books')
```

See also: *Managing users*.

### box.schema.user.revoke()

`box.schema.user.revoke(`*username*, *permissions*, *object-type*, *object-name*[, *{options}*]`)`
`box.schema.user.revoke(`*username*, *permissions*, *'universe'*[, *nil*, *{options}*]`)`
`box.schema.user.revoke(`*username*, *role-name*[, *nil*, *nil*, *{options}*]`)`

Отмена *прав* пользователя или другой роли.

**Параметры**

- `username` (*string*) – the name of the user
- `permissions` (*string*) – one or more *permissions* to revoke from the user (for example, `read` or `read,write`)
- `object-type` (*string*) – a database *object type* to revoke permissions from (for example, `space`, `role`, or `function`)
- `object-name` (*string*) – the name of a database object to revoke permissions from
- `options` (*table*) – `if_exists`

Должен существовать пользователь, должен существовать объект, но если задано `{if_exists=true}`, то ошибки не будет, если у пользователя нет прав.

**Вариант:** вместо `тип-объекта, имя-объекта` введите „universe“, что означает „все типы объектов и все объекты“.

**Variation:** instead of `permissions, object-type, object-name` say `role-name` (see section *Roles*).

**Variation:** instead of `box.schema.user.revoke('username','usage,session','universe',nil, {if_exists=true})` say `box.schema.user.disable('username')`.

**Пример:**

```
box.schema.user.revoke('testuser', 'write', 'space', 'books')
```

See also: *Managing users*.

### box.schema.user.password()

box.schema.user.password(*password*)

Выдача хеша пароля пользователя. Чтобы получить информацию о том, как происходит управление паролями в Tarantool, см. раздел *Пароли* и справочник по спейсу *_user*.

---

**Примечание:**

- Если у пользователя, который не является пользователем „guest“ нет пароля, **невозможно** подключиться к Tarantool через этого пользователя. Пользователь считается только “внутренним”, его нельзя использовать для удаленного подключения. Такие пользователи могут работать, если они определили какие-либо процедуры с помощью *SETUID*, на которые есть доступ у пользователей с внешним подключением. Таким образом, внешние пользователи могут не создавать/удалять объекты, а только вызывать процедуры.
- Для пользователя „guest“ невозможно установить пароль: это бы привело к путанице, поскольку „guest“ является пользователем по умолчанию для любого установленного подключения по *бинарному порту*, а Tarantool не требует пароль при установке *бинарного подключения*. Тем не менее, можно сменить текущего пользователя на пользователя ‘guest’, предоставив *AUTH-пакет* (пакет авторизации) без пароля или с пустым паролем. Данная функция полезна для пулов соединений, которые хотят повторно использовать соединение для другого пользователя без повторного подключения.

---

**Параметры**

- password (*string*) – пароль для хеширования

**тип возвращаемого значения** *string*

**Пример:**

```
box.schema.user.password('foobar')
```

### box.schema.user.passwd()

box.schema.user.passwd([*username*], *password*)

Sets a password for a currently logged in or a specified user:

- A currently logged-in user can change their password using `box.schema.user.passwd(password)`.
- An administrator can change the password of another user with `box.schema.user.passwd(username, password)`.

**Параметры**

- username (*string*) – a username
- password (*string*) – a new password

**Пример:**

```
box.schema.user.passwd('testuser', 'foobar')
```

See also: *Managing users*.

### box.schema.user.info()

box.schema.user.info([*username*])
Выдача описания *прав* пользователя.

**Параметры**

- username (*string*) – имя пользователя. Необязательный параметр; если не указать, информация будет для авторизованного пользователя.

See also: *Getting a user's information*.

### box.schema.role.create()

box.schema.role.create(*role-name*[, *{options}*])
Создание роли. Чтобы получить информацию о том, как происходит управление данными о ролях в Tarantool, см. раздел *Роли*.

**Параметры**

- role-name (*string*) – имя роли, которое должно соответствовать *правилам именования объектов*
- options (*table*) – if_not_exists (если отсутствует) = true|false (правда/ложь, по умолчанию ложь) - логическое значение boolean; true (правда) означает, что ошибка не выпадет, если роль уже существует.

**возвращает** nil

**Пример:**

```
box.schema.role.create('books_space_manager')
box.schema.role.create('writers_space_reader')
```

See also: *Managing roles*.

### box.schema.role.drop()

box.schema.role.drop(*role-name*[, *{options}*])
Удаление роли. Чтобы получить информацию о том, как происходит управление данными о ролях в Tarantool, см. раздел *Роли*.

**Параметры**

- role-name (*string*) – название роли
- options (*table*) – if_exists (если существует) = true|false (правда/ложь, по умолчанию ложь) - логическое значение boolean; true (правда) означает, что ошибка не выпадет, если такая роль не существует.

**Пример:**

```
box.schema.role.drop('writers_space_reader')
```

See also: *Managing roles*.

### box.schema.role.exists()

box.schema.role.exists(*role-name*)

Выдача `true` (правда), если роль существует; выдача `false` (ложь), если роль отсутствует.

**Параметры**

- `role-name` (*string*) – название роли

**тип возвращаемого значения** логическое значение bool

See also: *Getting a role's information*.

### box.schema.role.grant()

box.schema.role.grant(*role-name*, *permissions*, *object-type*, *object-name*[, *option*])

box.schema.role.grant(*role-name*, *permissions*, *'universe'*[, *nil*, *option*])

box.schema.role.grant(*role-name*, *role-name*[, *nil*, *nil*, *option*])

Выдача *прав* роли.

**Параметры**

- `role-name` (*string*) – the name of the role
- `permissions` (*string*) – one or more *permissions* to grant to the role (for example, `read` or `read,write`)
- `object-type` (*string*) – a database *object type* to grant permissions to (for example, `space`, `role`, or `function`)
- `object-name` (*string*) – the name of a database object to grant permissions to
- `option` (*table*) – `if_not_exists` = `true|false` (default = `false`) - boolean; `true` means there should be no error if the role already has the privilege

Должна существовать роль, должен существовать объект.

**Variation:** instead of `object-type, object-name` say `universe` which means „all object-types and all objects“. In this case, object name is omitted.

**Variation:** instead of `permissions, object-type, object-name` say `role-name` – to grant a role to a role.

**Пример:**

```
box.schema.role.grant('books_space_manager', 'read,write', 'space', 'books')
```

See also: *Managing roles*.

### box.schema.role.revoke()

box.schema.role.revoke(*role-name*, *permissions*, *object-type*, *object-name*)
Отмена *прав* роли.

**Параметры**

- role-name (*string*) – the name of the role
- permissions (*string*) – one or more *permissions* to revoke from the role (for example, read or read,write)
- object-type (*string*) – a database *object type* to revoke permissions from (for example, space, role, or function)
- object-name (*string*) – the name of a database object to revoke permissions from

Должна существовать роль, должен существовать объект, но ошибка не выпадет, если у роли нет прав.

**Variation:** instead of object-type, object-name say universe which means „all object-types and all objects“.

**Variation:** instead of permissions, object-type, object-name say role-name.

See also: *Managing roles*.

### box.schema.role.info()

box.schema.role.info(*role-name*)
Выдача описания прав роли.

**Параметры**

- role-name (*string*) – название роли.

See also: *Getting a role's information*.

### box.schema.func.create()

box.schema.func.create(*func_name*[, *function_options*])
Create a function. The created function can be used in different usage scenarios, for example, in *field or tuple constraints* or *functional indexes*.

Using the *body* option, you can make a function *persistent*. In this case, the function is «persistent» because its definition is stored in a snapshot (the *box.space._func* system space) and can be recovered if the server restarts.

**Параметры**

- func_name (*string*) – a name of the function, which should conform to the *rules for object names*
- function_options (*table*) – see *function_options*

**возвращает** nil

**Примечание:** *box.schema.user.grant()* can be used to allow the specified user or role to execute the created function.

### Example 1: a non-persistent Lua function

The example below shows how to create a non-persistent Lua function:

```
box.schema.func.create('calculate')
box.schema.func.create('calculate', {if_not_exists = false})
box.schema.func.create('calculate', {setuid = false})
box.schema.func.create('calculate', {language = 'LUA'})
```

### Example 2: a persistent Lua function

The example below shows how to create a persistent Lua function, show its definition using `box.func.{func-name}`, and call this function using `box.func.{func-name}:call([parameters])`:

```
tarantool> lua_code = [[function(a, b) return a + b end]]
tarantool> box.schema.func.create('sum', {body = lua_code})

tarantool> box.func.sum
---
- is_sandboxed: false
  is_deterministic: false
  id: 2
  setuid: false
  body: function(a, b) return a + b end
  name: sum
  language: LUA
...

tarantool> box.func.sum:call({1, 2})
---
- 3
...
```

To call functions using `net.box`, use *net_box:call()*.

### Example 3: a persistent SQL expression used in a tuple constraint

The code snippet below defines a function that checks a tuple's data using the SQL expression:

```
box.schema.func.create('check_person', {
    language = 'SQL_EXPR',
    is_deterministic = true,
    body = [["age" > 21 AND "name" != 'Admin']]
})
```

Then, this function is used to create a tuple *constraint*:

```
local customers = box.schema.space.create('customers', { constraint = 'check_person
↪' })
customers:format({
    { name = 'id', type = 'number' },
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    { name = 'name', type = 'string' },
    { name = 'age', type = 'number' },
})
customers:create_index('primary', { parts = { 1 } })
```

On an attempt to insert a tuple that doesn't meet the required criteria, an error is raised:

```
customers:insert { 2, "Bob", 18 }
-- error: Check constraint 'check_person' failed for a tuple
```

### function_options

`object function_options`

A table containing options passed to the *box.schema.func.create(func-name [, function_options])* function.

`function_options.if_not_exists`

Specify whether there should be no error if the function already exists.

Type: boolean

Default: `false`

`function_options.setuid`

Make Tarantool treat the function's caller as the function's creator, with full privileges. Note that `setuid` works only over *binary ports*. `setuid` doesn't work if you invoke a function using the *admin console* or inside a Lua script.

Type: boolean

Default: `false`

`function_options.language`

Specify the function language. The possible values are:

- `LUA`: define a Lua function in the *body* attribute.
- `SQL_EXPR`: define an *SQL expression* in the *body* attribute. An SQL expression can only be used as a field or tuple *constraint*.
- `C`: import a C function using its name from a `.so` file. Learn how to call C code from Lua in the *C tutorial*.

---

**Примечание:** To reload a C module with all its functions without restarting the server, call *box.schema.func.reload()*.

---

Type: string

Default: `LUA`

`function_options.is_sandboxed`
: Whether the function should be executed in an isolated environment. This means that any operation that accesses the world outside the sandbox is forbidden or has no effect. Therefore, a sandboxed function can only use modules and functions that cannot affect isolation:

  assert, assert, error, ipairs, math.*, next, pairs, pcall, print, select, string.*, table.*, tonumber, tostring, type, unpack, xpcall, *utf8.**.

  Also, a sandboxed function cannot refer to global variables – they are treated as local variables because the sandbox is established with setfenv. So, a sandboxed function is stateless and deterministic.

  Type: boolean

  Default: `false`

`function_options.is_deterministic`
: Specify whether a function should be deterministic.

  Type: boolean

  Default: `false`

`function_options.is_multikey`
: If `true` is set in the function definition for a functional index, the function returns multiple keys. For details, see the *example*.

  Type: boolean

  Default: `false`

`function_options.body`
: Specify a function body. You can set a function's language using the *language* attribute.

  The code snippet below defines a *constraint* function that checks a tuple's data using a Lua function:

```
box.schema.func.create('check_person', {
    language = 'LUA',
    is_deterministic = true,
    body = 'function(t, c) return (t.age >= 0 and #(t.name) > 3) end'
})
```

  In the following example, an SQL expression is used to check a tuple's data:

```
box.schema.func.create('check_person', {
    language = 'SQL_EXPR',
    is_deterministic = true,
    body = [["age" > 21 AND "name" != 'Admin']]
})
```

  Example: *A persistent SQL expression used in a tuple constraint*

Type: string
Default: `nil`

`function_options.takes_raw_args`
**Since:** *2.10.0*

If set to `true` for a Lua function and the function is called via `net.box` (*conn:call()*) or by `box.func.<func-name>:call()`, the function arguments are passed being wrapped in a *MsgPack object*:

```
local msgpack = require('msgpack')
box.schema.func.create('my_func', {takes_raw_args = true})
local my_func = function(mp)
    assert(msgpack.is_object(mp))
    local args = mp:decode() -- array of arguments
end
```

If a function forwards most of its arguments to another Tarantool instance or writes them to a database, the usage of this option can improve performance because it skips the MsgPack data decoding in Lua.

Type: boolean
Default: `false`

`function_options.exports`
Specify the languages that can call the function.

Example: `exports = {'LUA', 'SQL'}`

See also: *Calling Lua routines from SQL*

Type: table
Default: `{'LUA'}`

`function_options.param_list`
Specify the Lua type names for each parameter of the function.

Example: `param_list = {'number', 'number'}`

See also: *Calling Lua routines from SQL*

Type: table

`function_options.returns`
Specify the Lua type name for a function's return value.

Example: `returns = 'number'`

See also: *Calling Lua routines from SQL*

Type: string

### box.schema.func.drop()

box.schema.func.drop(*func-name*[, *{options}*])
Удаление кортежа с функцией. Чтобы получить информацию о том, как происходит управление данными функций в Tarantool, см. справочник по спейсу *_func*.

**Параметры**

- func-name (*string*) – имя функции
- options (*table*) – if_exists (если существует) = true|false (правда/ложь, по умолчанию ложь) - логическое значение boolean; true (правда) означает, что ошибка не выпадет, если кортеж в _func не существует.

**Пример:**

```
box.schema.func.drop('calculate')
```

### box.schema.func.exists()

box.schema.func.exists(*func-name*)
Выдача true (правда), если кортеж с функцией существует; выдача false (ложь), если кортеж с функцией отсутствует.

**Параметры**

- func-name (*string*) – имя функции

**тип возвращаемого значения** логическое значение bool

**Пример:**

```
box.schema.func.exists('calculate')
```

### box.schema.func.reload()

box.schema.func.reload([*name*])
Перезагрузка модуля на C (со всеми его функциями) без перезапуска сервера.

С точки зрения внутреннего устройства, Tarantool загружает новую копию модуля (библиотека общего пользования *.so) и запускает маршрутизацию всех новых запросов на новую версию. Предыдущая версия остается активной до тех пор, пока не завершатся все начатые вызовы. Все библиотеки общего пользования загружены с RTLD_LOCAL (см. «man 3 dlopen»), таким образом, множество копий могут работать одновременно без каких-либо проблем.

---

**Примечание:** Перезагрузка не сработает, если модуль был загружен из Lua-скрипта с ffi.load().

---

**Параметры**

- name (*string*) – имя модуля для перезагрузки

**Пример:**

```
-- перегрузить целиком всё содержимое модуля
box.schema.func.reload('module')
```

### Последовательности

Вводная информация о последовательностях дается в разделе *Последовательности* главы «Модель данных». Здесь же приведена подробная информация о каждой функции и каждом параметре.

Все функции, связанные с последовательностями, требуют наличия соответствующих *прав*.

Ниже приведен перечень всех функций модуля `box.schema.sequence`.

| Имя | Использование |
|---|---|
| *box.schema.sequence.create()* | Создание нового генератора последовательностей |
| *sequence_object:next()* | Генерация и выдача следующего значения |
| *sequence_object:alter()* | Изменение параметров последовательности |
| *sequence_object:reset()* | Возврат последовательности в оригинальное состояние |
| *sequence_object:set()* | Установка нового значения |
| *sequence_object:current()* | Возврат последнего найденного значения |
| *sequence_object:drop()* | Удаление последовательности |
| *использование последовательностей в create_index()* | Создание индекса с опцией последовательности |

### box.schema.sequence.create()

`box.schema.sequence.create`(*name*[, *options*])

Создание нового генератора последовательностей.

**Параметры**

- `name` (*string*) – имя последовательности
- `options` (*table*) – см. краткий обзор в *таблице* «Параметры для `box.schema.sequence.create()`» (в разделе *Последовательности* главы «Модель данных»), а более подробную информацию ниже.

**возвращает** ссылка на новый объект последовательности.

Параметры:

- `start` – НАЧАЛЬНОЕ значение. Тип = целое число, по умолчанию = 1.
- `min` – МИНИМАЛЬНОЕ значение. Тип = целое число, по умолчанию = 1.
- `max` –МАКСИМАЛЬНОЕ значение. Тип = целое число, по умолчанию = 9223372036854775807.

  Есть следующее правило: `min` <= `start` <= `max`. Например, нельзя указать `{start=0}`, поскольку указанное начальное значение (0) будет меньше, чем минимальное значение, используемое по умолчанию (1).

  Есть следующее правило: `min` <= следующее-значение <= `max`. Например, если сгенерированное значение будет 1000, но максимальное значение – 999, это будет считаться переполнением.

There is a rule: `start` and `min` and `max` must all be <= 9223372036854775807 which is 2^63 - 1 (not 2^64).

- `cycle` – значение ЦИКЛА. Тип = bool (логический), по умолчанию = false (ложь).

  Если следующее значение в генераторе последовательности будет переполнением, это вызовет ошибку – не считая случаев, когда задан цикл (`cycle == true`).

  Если же `cycle == true`, отсчет начинается заново с МИНИМАЛЬНОГО значения или с МАКСИМАЛЬНОГО значения (не с НАЧАЛЬНОГО значения).

- `cache` – значение КЭША. Тип = беззнаковое целое число, по умолчанию = 0.

  В данный момент Tarantool игнорирует это значение, оно зарезервировано для последующего использования.

- `step` – значение УВЕЛИЧЕНИЯ. Тип = целое число, по умолчанию = 1.

  Это значение прибавляется к предыдущему.

### sequence_object:next()

```
object sequence_object
```

`sequence_object:next()`

Генерация и выдача следующего значения.

Простой алгоритм для генерации:

- В первый раз вернуть НАЧАЛЬНОЕ значение.
- Если предыдущее значение плюс значение УВЕЛИЧЕНИЯ меньше, чем МИНИМАЛЬНОЕ значение, или больше, чем МАКСИМАЛЬНОЕ значение, будет переполнение, поэтому либо выдать сообщение об ошибке (если цикл не задан – `cycle = false`) или вернуть МАКСИМАЛЬНОЕ значение (если цикл задан – `cycle = true` – и `step` < 0), или вернуть МИНИМАЛЬНОЕ значение (если цикл задан – `cycle = true` – и `step` > 0).

Если ошибки нет, сохранить результат, который становится «предыдущим значением».

Например, предположим, что для последовательности „S“:

- `min` == -6,
- `max` == -1,
- `step` == -3,
- `start` = -2,
- `cycle` = true,
- предыдущее значение = -2.

Тогда `box.sequence.S:next()` вернет -5, потому что -2 + (-3) == -5.

Затем `box.sequence.S:next()` снова вернет -1, потому что -5 + (-3) < -6, что будет переполнением, которое вызовет цикл, а `max` == -1.

Для данной функции необходимы права на *запись („write“)* на последовательность.

**Примечание:** Данную функцию не следует использовать в транзакциях между движками (транзакции, в которых используется и движок memtx, и движок vinyl).

Чтобы увидеть предыдущее значение, не изменяя его, сделайте выборку из системного спейса *_sequence_data*.

### sequence_object:alter()

```
object sequence_object
```

sequence_object:alter(*options*)

Функцию `alter()` можно использовать для изменения любых параметров последовательности. Требования и ограничения в данном случае такие же, как для *box.schema.sequence.create()*.

Параметры:

- `start` – НАЧАЛЬНОЕ значение. Тип = целое число, по умолчанию = 1.
- `min` – МИНИМАЛЬНОЕ значение. Тип = целое число, по умолчанию = 1.
- `max` –МАКСИМАЛЬНОЕ значение. Тип = целое число, по умолчанию = 9223372036854775807.

  Есть следующее правило: `min` <= `start` <= `max`. Например, нельзя указать `{start=0}`, поскольку указанное начальное значение (0) будет меньше, чем минимальное значение, используемое по умолчанию (1).

  Есть следующее правило: `min` <= следующее-значение <= `max`. Например, если сгенерированное значение будет 1000, но максимальное значение – 999, это будет считаться переполнением.
- `cycle` – значение ЦИКЛА. Тип = bool (логический), по умолчанию = false (ложь).

  Если следующее значение в генераторе последовательности будет переполнением, это вызовет ошибку – не считая случаев, когда задан цикл (`cycle == true`).

  Если же `cycle == true`, отсчет начинается заново с МИНИМАЛЬНОГО значения или с МАКСИМАЛЬНОГО значения (не с НАЧАЛЬНОГО значения).
- `cache` – значение КЭША. Тип = беззнаковое целое число, по умолчанию = 0.

  В данный момент Tarantool игнорирует это значение, оно зарезервировано для последующего использования.
- `step` – значение УВЕЛИЧЕНИЯ. Тип = целое число, по умолчанию = 1.

  Это значение прибавляется к предыдущему.

### sequence_object:reset()

```
object sequence_object
```

sequence_object:reset()
Возврат последовательности в оригинальное состояние. Смысл в том, что последующий вызов `next()` вернет начальное значение `start`. Для данной функции необходимы права на *запись („write“)* на последовательность.

### sequence_object:set()

```
object sequence_object
```

sequence_object:set(*new-previous-value*)
Установка «предыдущего значения» на `new-previous-value` (новое предыдущее значение). Для данной функции необходимы права на *запись („write“)* на последовательность.

### sequence_object:current()

```
object sequence_object
```

sequence_object:current()
Since version *2.4.1*. Return the last retrieved value of the specified sequence or throw an error if no value has been generated yet (`next()` has not been called yet, or `current()` is called right after `reset()` is called).

**Пример:**

```
tarantool> sq = box.schema.sequence.create('test')
---
...
tarantool> sq:current()
---
- error: Sequence 'test' is not started
...
tarantool> sq:next()
---
- 1
...
tarantool> sq:current()
---
- 1
...
tarantool> sq:set(42)
---
...
tarantool> sq:current()
---
- 42
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> sq:reset()
---
...
tarantool> sq:current()  -- error
---
- error: Sequence 'test' is not started
...
```

### sequence_object:drop()

```
object sequence_object
```

sequence_object:drop()
Удаление существующей последовательности.

### использование последовательностей в create_index()

```
object space_object
```

space_object:create_index(... *[sequence='...' option]* ...)
Можно использовать опцию sequence=*имя-последовательности* (или sequence=*id-последовательности*, или sequence=true) при *создании* или *изменении* первичного индекса. Происходит ассоциация последовательности с индексом, так что следующий вызов insert() поместит следующее сгенерированное число в поле первичного ключа, если в противном случае поле было бы nil.

The syntax may be any of: sequence = *sequence identifier* or sequence = {id = *sequence identifier* } or sequence = {field = *field number* } or sequence = {id = *sequence identifier* , field = *field number* } or sequence = true or sequence = {}. The sequence identifier may be either a number (the sequence id) or a string (the sequence name). The field number may be the ordinal number of any field in the index; default = 1. Examples of all possibilities: sequence = 1 or sequence = 'sequence_name' or sequence = {id = 1} or sequence = {id = 'sequence_name'} or sequence = {id = 1, field = 1} or sequence = {id = 'sequence_name', field = 1} or sequence = {field = 1} or sequence = true or sequence = {}. Notice that the sequence identifier can be omitted, if it is omitted then a new sequence is created automatically with default name = *space-name*_seq. Notice that the field number does not have to be 1, that is, the sequence can be associated with any field in the primary-key index.

Например, если „Q“ – это последовательность, а „T“ – это новый спейс, то сработает:

```
tarantool> box.space.T:create_index('Q',{sequence='Q'})
---
- unique: true
  parts:
  - type: unsigned
    is_nullable: false
    fieldno: 1
  sequence_id: 8
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  id: 0
  space_id: 514
  name: Q
  type: TREE
...
```

(Обратите внимание, что теперь в индексе есть поле идентификатора последовательности `sequence_id`.)

И сработает:

```
tarantool> box.space.T:insert{box.NULL,0}
---
- [1, 0]
...
```

---

**Примечание:** The index key type may be either „integer“ or „unsigned“. If any of the sequence options is a negative number, then the index key type should be „integer“.

Users should not insert a value greater than 9223372036854775807, which is 2^63 - 1, in the indexed field. The sequence generator will ignore it.

Последовательность нельзя удалить, если она связана с индексом. Тем не менее, можно использовать *index_ object:alter()*, чтобы показать, что последовательность не связана с индексом, например так `box.space.T.index.I:alter({sequence=false})`.

If a sequence was created automatically because the sequence identifier was omitted, then it will be dropped automatically if the index is altered so that `sequence=false`, or if the index is dropped.

`index_object:alter()` can also be used to associate a sequence with an existing index, with the same syntax for options.

When a sequence is used with an index based on a JSON path, inserted tuples must have all components of the path preceding the autoincrement field, and the autoincrement field. To achieve that use `box.NULL` rather than `nil`. Example:

```
s = box.schema.space.create('test')
s:create_index('pk', {parts = {{'[1].a.b[1]', 'unsigned'}}, sequence = true})
s:replace{} -- error
s:replace{{c = {}}} -- error
s:replace{{a = {c = {}}}} -- error
s:replace{{a = {b = {}}}} -- error
s:replace{{a = {b = {nil}}}} -- error
s:replace{{a = {b = {box.NULL}}}} -- ok
```

---

**Пример:**

Ниже представлен пример, иллюстрирующий все параметры и операции для последовательностей:

```
s = box.schema.sequence.create(
                'S2',
                {start=100,
                min=100,
                max=200,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                cache=100000,
                cycle=false,
                step=100
                })
s:alter({step=6})
s:next()
s:reset()
s:set(150)
s:drop()
```

### Вложенный модуль box.session

Вложенный модуль `box.session` позволяет делать запросы состояния сессии, вносить записи во временную Lua-таблицу по отдельной сессии, отправлять экстренные сообщения и настраивать триггеры, которые сработают в начале или окончании сессии.

*Сессия* – это объект, связанный с каждым подключением клиента.

Ниже приведен перечень всех функций и элементов модуля `box.session`.

| Имя | Использование |
|---|---|
| *box.session.id()* | Получение идентификатора текущей сессии |
| *box.session.exists()* | Проверка наличия сессии |
| *box.session.peer()* | Получение адреса хоста и порта подключенного узла |
| *box.session.sync()* | Получение целочисленной константы sync |
| *box.session.user()* | Получение имени текущего пользователя |
| *box.session.type()* | Получение типа соединения или повода к действию |
| *box.session.su()* | Изменение текущего пользователя |
| *box.session.uid()* | Получение идентификатора текущего пользователя |
| *box.session.euid()* | Получение идентификатора текущего действующего пользователя |
| *box.session.storage* | Таблица с именами и значениями по сессии |
| *box.session.on_connect()* | Определение триггера для подключения |
| *box.session.on_disconnect()* | Определение триггера для отключения |
| *box.session.on_auth()* | Определение триггера для аутентификации |
| *box.session.on_access_denied()* | Определение триггера для регистрации запрещенных действий |
| *box.session.push()* | Отправка внеполосного сообщения |

### box.session.id()

```
box.session.id()
```

Return the unique identifier (ID) for the current session.

**возвращает** the session identifier; 0 or -1 if there is no session

**тип возвращаемого значения** число

### box.session.exists()

box.session.exists(*id*)

**возвращает** 1, если сессия есть; 0, если сессии нет.

**тип возвращаемого значения** boolean (логический)

### box.session.peer()

box.session.peer(*id*)

Данная функция сработает только в том случае, если есть подключенная программа, то есть если было выполнено подключение к отдельному экземпляру Tarantool.

**возвращает** Адрес хоста и порт подключенного узла, например «127.0.0.1:55457». Если существует сессия, но отсутствует подключение к отдельному экземпляру, вернется null. Команда выполняется на экземпляре сервера, поэтому «локальное имя» – это хост и порт экземпляра сервера, а «имя узла» – это хост и порт клиента.

**тип возвращаемого значения** string (строка)

Возможные ошибки: „session.peer(): сессия отсутствует“

### box.session.sync()

box.session.sync()

**возвращает** значение целочисленной константы `sync`, используемой в бинарном протоколе. Это значение будет недействительным после отключения сессии.

**тип возвращаемого значения** число

This function is local for the request, i.e. not global for the session. If the connection behind the session is multiplexed, this function can be safely used inside the request processor.

### box.session.user()

box.session.user()

**возвращает** имя *текущего пользователя*

**тип возвращаемого значения** строка

### box.session.type()

box.session.type()

**возвращает** тип соединения или повод к действию.

**тип возвращаемого значения** *string*

Возможные возвращаемые значения:

- „binary“ (бинарное), если подключение было выполнено по бинарному протоколу, например, к объекту с помощью *box.cfg{listen=...}*;
- „console“ (консоль), если подключение было выполнено по административной консоли, например, к объекту с помощью *console.listen*;
- „repl“ (репликация), если подключение было выполнено напрямую, например, при *использовании Tarantool в качестве клиента*;
- „applier“ (наложение), если действие происходит по причине *репликации*, независимо от типа подключения;
- „background“ (в фоне), если действие происходит в *фоновом файбере*, независимо от того, был ли Tarantool *запущен в фоновом режиме*.

`box.session.type()` используется для триггера при замене *on_replace()* на реплике – значение будет „applier“ только в том случае, если триггер был активирован по причине запроса, выполненного на мастере.

### box.session.su()

box.session.su(*user-name*[, *function-to-execute*])

Изменение *текущего пользователя* Tarantool – аналогично Unix-команде `su`.

Или, если указана выполняемая функция (function-to-execute), временное изменение *текущего пользователя* Tarantool во время выполнения функции – аналогично Unix-команде `sudo`.

**Параметры**

- `user-name` (*string*) – целевое имя пользователя
- `function-to-execute` – имя функции или определение функции. Дополнительные параметры могут передаваться в `box.session.su`, они будут интерпретироваться как параметры выполняемой функции.

**Пример:**

```
tarantool> function f(a) return box.session.user() .. a end
---
...

tarantool> box.session.su('guest', f, '-xxx')
---
- guest-xxx
...

tarantool> box.session.su('guest',function(...) return ... end,1,2)
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- 1
- 2
...
```

### box.session.uid()

```
box.session.uid()
```

**возвращает** ID *текущего пользователя*.

**тип возвращаемого значения** число

У каждого пользователя есть уникальное имя (узнать с помощью *box.session.user()*) и уникальный идентификатор (узнать с помощью `box.session.uid()`). Значения хранятся вместе в спейсе `_user`.

### box.session.euid()

```
box.session.euid()
```

**возвращает** рабочий ID *текущего пользователя*.

Аналогично *box.session.uid()*, за исключением двух случаев:

- Первый случай: если вызов `box.session.euid()` выполняется в рамках функции, вызываемой по *box.session.su(user-name, function-to-execute)* – в таком случае `box.session.euid()` вернет измененный идентификатор пользователя (пользователь, который указан в параметре `user-name` функции `su`), но `box.session.uid()` вернет идентификатор оригинального пользователя (пользователя, который вызывает функцию `su`).
- Второй случай: если вызов `box.session.euid()` выполняется в рамках функции по *box.schema.func.create(function-name, {setuid= true})*, и используется бинарный протокол – в таком случае `box.session.euid()` вернет идентификатор пользователя, который создал функцию «function-name», а `box.session.uid()` вернет идентификатор пользователя, который вызывает эту функцию «function-name».

**тип возвращаемого значения** число

**Пример:**

```
tarantool> box.session.su('admin')
---
...
tarantool> box.session.uid(), box.session.euid()
---
- 1
- 1
...
tarantool> function f() return {box.session.uid(),box.session.euid()} end
---
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> box.session.su('guest', f)
---
- - 1
  - 0
...
```

### box.session.storage

box.session.storage
: Lua-таблица с произвольными неупорядоченными именами и значениями по сессии, которая хранится до конца сессии. Например, эту таблицу можно использовать для хранения текущих задач при работе с очередями сообщений в Tarantool.

**Пример:**

```
tarantool> box.session.peer(box.session.id())
---
- 127.0.0.1:45129
...
tarantool> box.session.storage.random_memorandum = "Don't forget the eggs"
---
...
tarantool> box.session.storage.radius_of_mars = 3396
---
...
tarantool> m = ''
---
...
tarantool> for k, v in pairs(box.session.storage) do
         >   m = m .. k .. '='.. v .. ' '
         > end
---
...
tarantool> m
---
- 'radius_of_mars=3396 random_memorandum=Don't forget the eggs. '
...
```

### box.session.on_connect()

box.session.on_connect(*trigger-function*[, *old-trigger-function*])
: Определение исполняемого триггера во время создания новой сессии при подключении по консоли *console.connect*. Функция с триггером будет первой исполняемой функцией после создания сессии. Если триггер не выполняется и выдает ошибку, эта ошибка отправляется на клиент, и подключение разрывается.

**Параметры**

- `trigger-function` (`function`) – функция, в которой будет триггер
- `old-trigger-function` (`function`) – существующая функция с триггером, которую заменит новая

**возвращает** nil или указатель функции

Если указаны параметры (nil, old-trigger-function), старый триггер будет удален.

Если не указан ни один параметр, ответом будет список существующих функций с триггером.

Подробная информация о характеристиках триггера находится в разделе *Триггеры*.

**Пример:**

```
tarantool> function f ()
         >   x = x + 1
         > end
tarantool> box.session.on_connect(f)
```

**Предупреждение:** Если триггер всегда приводит к ошибке, подключение к серверу для его переустановки может стать невозможным.

### box.session.on_disconnect()

box.session.on_disconnect([*trigger-function*[, *old-trigger-function*]])

Определение исполняемого триггера после отключения клиента. Если функция с триггером вызывает ошибку, то ошибка записывается в журнал, в противном случае записей не будет. Триггер вызывается во время сессии клиента и может получить доступ к свойствам сессии, как *box.session.id()*.

Начиная с версии 1.10, функция с триггером вызывается сразу же после прерывания сессии, даже если сделанные запросы не были выполнены.

**Параметры**

- `trigger-function` (`function`) – функция, в которой будет триггер
- `old-trigger-function` (`function`) – существующая функция с триггером, которую заменит новая

**возвращает** nil или указатель функции

Если указаны параметры (nil, old-trigger-function), старый триггер будет удален.

Если не указан ни один параметр, ответом будет список существующих функций с триггером.

Подробная информация о характеристиках триггера находится в разделе *Триггеры*.

**Пример №1**

```
tarantool> function f ()
         >   x = x + 1
         > end
tarantool> box.session.on_disconnect(f)
```

**Пример №2**

После следующей серии запросов экземпляр Tarantool запишет сообщение с помощью модуля *log* при подключении или отключении любого пользователя.

```
function log_connect ()
  local log = require('log')
  local m = 'Connection. user=' .. box.session.user() .. ' id=' .. box.session.id()
  log.info(m)
end

function log_disconnect ()
  local log = require('log')
  local m = 'Disconnection. user=' .. box.session.user() .. ' id=' .. box.session.
↪id()
  log.info(m)
end

box.session.on_connect(log_connect)
box.session.on_disconnect(log_disconnect)
```

Вот что может быть записано в файл журнала при обычной установке:

```
2014-12-15 13:21:34.444 [11360] main/103/iproto I>
    Connection. user=guest id=3
2014-12-15 13:22:19.289 [11360] main/103/iproto I>
    Disconnection. user=guest id=3
```

### box.session.on_auth()

box.session.on_auth([*trigger-function*[, *old-trigger-function*]])

Определение триггера, используемого во время *аутентификации*.

Вызов функции `on_auth` с триггером происходит в следующих обстоятельствах:

(1) Функция *console.connect* включает в себя проверку аутентификации всех пользователей, кроме „guest“. Вызов функции `on_auth` с триггером происходит после триггера `on_connect` только в том случае, если подключение было успешным.

(2) В *бинарном протоколе* есть отдельный *пакет для аутентификации*. В этом случае подключение и аутентификация считаются отдельными действиям.

В отличие от других типов триггеров, вызов функций с триггером `on_auth` происходит **до** события. Таким образом, функция с таким триггером, как `function auth_function () v = box.session.user(); end`, определит `v` как «guest», то есть имя пользователя до проведения аутентификации. Чтобы получить имя пользователя **после** проведения аутентификации, используйте специальный синтаксис: `function auth_function (user_name) v = user_name; end`

Если триггер не выполняется и выдает ошибку, эта ошибка отправляется на клиент, и подключение разрывается.

**Параметры**

- `trigger-function` (`function`) – функция, в которой будет триггер
- `old-trigger-function` (`function`) – существующая функция с триггером, которую заменит новая

**возвращает** nil или указатель функции

Если указаны параметры (nil, old-trigger-function), старый триггер будет удален.

Если не указан ни один параметр, ответом будет список существующих функций с триггером.

Подробная информация о характеристиках триггера находится в разделе *Триггеры*.

**Пример 1**

```
tarantool> function f ()
         >   x = x + 1
         > end
tarantool> box.session.on_auth(f)
```

**Пример 2**

Более сложный пример с двумя экземплярами сервера.

Первый экземпляр сервера настроен на прослушивание по порту 3301; имя пользователя по умолчанию – „admin“. Есть три триггера `on_auth`:

- В первом триггере есть функция без аргументов, которая только смотрит на `box.session.user()`.
- Во втором триггере есть функция с аргументом `user_name`, которая может смотреть на `box.session.user()` и `user_name`.
- В третьем триггере есть функция с аргументом `user_name` и аргументом `status`, которая может смотреть на `box.session.user()` и `user_name`, и``status``.

Второй экземпляр сервера подключится по *console.connect*, а затем отобразит переменные, определенные функциями с триггером.

```
-- На первом экземпляре сервера, прослушивание на котором настроено на порт 3301
box.cfg{listen=3301}
function function1()
  print('function 1, box.session.user()='..box.session.user())
  end
function function2(user_name)
  print('function 2, box.session.user()='..box.session.user())
  print('function 2, user_name='..user_name)
  end
function function3(user_name, status)
  print('function 3, box.session.user()='..box.session.user())
  print('function 3, user_name='..user_name)
  if status == true then
    print('function 3, status = true, authorization succeeded')
    end
  end
box.session.on_auth(function1)
box.session.on_auth(function2)
box.session.on_auth(function3)
box.schema.user.passwd('admin')
```

```
-- На втором экземпляре сервера, который подключается по порту 3301
console = require('console')
console.connect('admin:admin@localhost:3301')
```

Теперь результат выглядит следующим образом:

```
function 3, box.session.user()=guest
function 3, user_name=admin
function 3, status = true, authorization succeeded
function 2, box.session.user()=guest
function 2, user_name=admin
function 1, box.session.user()=guest
```

### box.session.on_access_denied()

box.session.on_access_denied([*trigger-function*[, *old-trigger-function*]])

Определение триггера для ответа на попытки пользователя выполнить неразрешенные ему действия.

**Параметры**

- `trigger-function` (`function`) – функция, в которой будет триггер
- `old-trigger-function` (`function`) – существующая функция с триггером, которую заменит новая

**возвращает** nil или указатель функции

Если указаны параметры `(nil, old-trigger-function)`, старый триггер будет удален.

Если не указан ни один параметр, ответом будет список существующих функций с триггером.

Подробная информация о характеристиках триггера находится в разделе *Триггеры*.

**Пример:**

Например, администратор сервера может регистрировать запрещенные действия:

```
tarantool> function on_access_denied(op, type, name)
         > log.warn('User %s tried to %s %s %s without required privileges', box.
↪session.user(), op, type, name)
         > end
---
...
tarantool> box.session.on_access_denied(on_access_denied)
---
- 'function: 0x011b41af38'
...
tarantool> function test() print('you shall not pass') end
---
...
tarantool> box.schema.func.create('test')
---
...
```

И когда какой-нибудь пользователь без соответствующих прав попытается вызвать `test()`` и получит ошибку, сервер выполнит этот триггер и запишет в журнал **«User *имя_пользователя* tried to Execute function test without required privileges»** (Пользователь *имя_пользователя* пытался выполнить функцию текст без соответствующих прав).

### box.session.push()

box.session.push(*message*[, *sync*])

Создание внеполосного сообщения. Под внеполосным мы понимаем дополнительное сообщение, которое дополняет то, что отправляется в сети по обычным каналам. Хотя `box.session.push()` можно вызвать в любое время, на практике эта функция используется в сетях, настроенных с помощью *модуля net.box*, и вызывается сервером (на «удаленной системе с базой данных», если использовать нашу терминологию для net.box), а у клиента есть возможность принимать такие сообщения.

Функция возвращает ошибку, если сессия была прервана.

**Параметры**

- `message` (`any-Lua-type`) – что отправляется
- `sync` (`int`) – необязательный аргумент, указывающий на сессию. Этот аргумент берётся из предшествующего вызова *box.session.sync()*. Если аргумент опущен, применяется значение по умолчанию — текущее значение `box.session.sync()`. Аргумент признан устаревшим в версии Tarantool *2.4.2*, а начиная с версии 2.5.1 его использование приводит к ошибке.

**тип возвращаемого значения** {nil, ошибка} или true:

- Если результатом будет ошибка, то вернется `nil` вместе с объектом ошибки.
- Если результатом будет не ошибка, то вернется логическое значение `true` (правда).
- When the return is `true`, the message has gone to the network buffer as a *packet* with a different header code so the client can distinguish from an ordinary Okay response.

Единственная задача сервера – вызвать `box.session.push()`, поскольку нет автоматического механизма, который показал бы, что сообщение получено.

Задача клиента заключается в том, чтобы проверять наличие таких сообщений после отправки чего-либо на сервер. Основные клиентские методы – *conn:call*, *conn:eval*, *conn:select*, *conn:insert*, *conn:replace*, *conn:update*, *conn:upsert*, *delete* – могут привести к отправке такого сообщения сервером.

Ситуация 1: когда клиент делает синхронный вызов со значением параметра `{async=false}` по умолчанию. Есть два необязательных дополнительных параметра: `on_push=`*`function-name`* и `on_push_ctx=`*`function-argument`*. Когда клиент получает внеполосное сообщение в сессии, он вызывает «имя-функции(аргумент-функции)». Например, с такими значениями параметров: `{on_push=table.insert, on_push_ctx=messages}` – клиент произведет вставку полученных данных в таблицу под названием „messages“.

Ситуация 2: когда клиент делает асинхронный вызов с измененным значением параметра `{async=true}`. Здесь не разрешены `on_push` и `on_push_ctx`, но сообщения можно увидеть путем вызова `pairs()` в цикле.

Осложненная ситуация 2: `pairs()` зависит от времени ожидания. Таким образом, есть необязательный аргумент – время ожидания для итерации. Если время ожидания истечет до получения нового сообщения или окончательного ответа, вернется ошибка. Чтобы проверить наличие ошибки, можно использовать первый параметр в цикле (если цикл начинается с «for i, message in future:pairs()», то первым параметром в цикле будет i). Если это будет `box.NULL`, то второй параметр (в нашем примере «message») – это объект ошибки.

**Пример:**

```
-- Создайте две оболочки. В оболочке №1 настройте сервер, а
-- в нем функцию, которая содержит box.session.push:
box.cfg{listen=3301}
box.schema.user.grant('guest','read,write,execute','universe')
x = 0
fiber = require('fiber')
function server_function() x=x+1; fiber.sleep(1); box.session.push(x); end

-- В оболочке №2 подключитесь к серверу в качестве клиента, который
-- поддерживает Lua (как второй Tarantool-сервер, работающий
-- в качестве клиента), и создайте таблицу, в которую мы будем получать сообщения:
net_box = require('net.box')
conn = net_box.connect(3301)
messages_from_server = {}

-- В оболочке №2 удаленно вызовите функцию и получите
-- СИНХРОННОЕ внеполосное сообщение:
conn:call('server_function', {},
          {is_async = false,
           on_push = table.insert,
           on_push_ctx = messages_from_server})
messages_from_server
-- Через секунду, во время которой происходит запрос fiber.sleep()
-- в server_function, результат в таблице
--  messages_from_server будет следующим: 1. Проверим:
-- tarantool> messages_from_server
-- ---
-- - - 1
-- ...
-- Хорошо. Это означает, что box.session.push(x) сработала,
-- поскольку мы знаем, что x был 1.

-- В оболочке №2 удаленно вызовите ту же самую функцию
-- для получения АСИНХРОННОГО внеполосного сообщения. При этом мы не можем
-- использовать параметры on_push и on_push_ctx, но можем использовать pairs():
future = conn:call('server_function', {}, {is_async = true})
messages = {}
keys = {}
for i, message in future:pairs() do
    table.insert(messages, message) table.insert(keys, i) end
messages
future:wait_result(1000)
for i, message in future:pairs() do
    table.insert(messages, message) table.insert(keys, i) end
messages
-- Задержки нет, поскольку conn:call не ждет
-- окончания вызова функции server_function. После первой итерации
-- цикла pairs(), видим, что таблица пуста. Это выглядит так:
-- tarantool> messages
-- ---
-- - - 2
--   - []
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- ...
-- Это нормально, поскольку сервер еще не вызвал
-- box.session.push(). При второй итерации
-- цикла pairs(), видим значение x во время
-- второго вызова box.session.push(). Так:
-- tarantool> messages
-- ---
-- - - 2
--   - &0 []
--   - 2
--   - *0
-- ...
-- Хорошо. Это означает, что сообщение было асинхронным, и
-- box.session.push() выполнила свою задачу.
```

### Вложенный модуль box.slab

Вложенный модуль `box.slab` предоставляет доступ к статистике распределения slab. Механизм распределения slab представляет собой основной тип распределения для хранения *кортежей*. Такое распределение можно использовать для отслеживания использования памяти и фрагментации памяти.

Ниже приведен перечень всех функций модуля `box.slab`.

| Имя | Использование |
|---|---|
| *box.runtime.info()* | Отображение отчета по использованию памяти во время исполнения Lua-кода |
| *box.slab.info()* | Отображение обобщенного отчета по использованию памяти для распределения slab |
| *box.slab.stats()* | Отображение подробного отчета по использованию памяти для распределения slab |

### box.runtime.info()

`box.runtime.info()`

Отображение отчета по использованию runtime-памяти в байтах.

Runtime-память включает в себя внутреннюю память Lua и runtime-арену. В памяти Lua хранятся объекты Lua. На runtime-арене хранятся объекты, специфичные для Tarantool: временные пользовательские кортежи, сетевые буферы и прочие объекты, связанные с подсистемой сервера приложений.

**возвращает**

- `lua` – размер динамической памяти Lua, контролируемой сборщиком мусора в Lua;
- `maxalloc` – максимальный размер runtime-памяти;
- `used` – объем памяти, используемый runtime-памятью в данный момент.

**тип возвращаемого значения** таблица

**Пример:**

```
tarantool> box.runtime.info()
---
- lua: 913710
  maxalloc: 4398046510080
  used: 12582912
...
tarantool> box.runtime.info().used
---
- used: 12582912
...
```

### box.slab.info()

box.slab.info()

Show an aggregated memory usage report in bytes for the slab allocator. This report is useful for assessing out-of-memory risks.

`box.slab.info` выдает несколько показателей:

- items_used_ratio
- arena_used_ratio
- quota_used_ratio

При мониторинге используемой памяти в memtx'е есть два возможных сценария:

**1 сценарий:** $0.5 <$ `items_used_ratio` $< 0.9$

Очевидно, память сильно фрагментирована. Проверьте, сколько у вас классов slab, подсчитав количество различных классов с помощью `box.slab.stats()`. Если классов slab много (больше нескольких десятков), то память может закончиться, даже если её занято не так много. На каждом slab может быть использовано мало элементов. Но всякий раз при выделении кортежа, размер которого отличается от любого существующего класса, Tarantool может понадобиться новый slab из области распределения slab. И если осталось мало пустых slab, то произойдет попытка увеличения квоты, что, в свою очередь, может привести к ошибке нехватки памяти из-за низкой оставшейся квоты памяти.

**2 сценарий:** `items_used_ratio` $> 0.9$

Память заканчивается. Высокие показатели использования памяти. Память не фрагментирована, но каждый уровень механизма распределения slab почти пуст. Следует подумать об увеличении лимита памяти Tarantool (`box.cfg.memtx_memory`).

**Вывод:** основной показатель нехватки памяти – `quota_used_ratio`. Тем не менее, существует множество абсолютно стабильных установок с высоким показателем `quota_used_ratio` , поэтому необходимо обращать на это внимание, когда два других показателя также высоки (arena и item used).

**возвращает**

- `quota_size` – лимит памяти для механизма распределения slab (как настроено в параметре *memtx_memory*, по умолчанию 2^28 байтов = 268 435 456 байтов)
- `quota_used` – использовано механизмом распределения slab

- `items_size` – выделено только для кортежей
- `items_used` – использовано только для кортежей
- `arena_size` – выделено для кортежей и индексов вместе
- `arena_used` – использовано для кортежей и индексов вместе
- `items_used_ratio` = `items_used` / `items_size`
- `quota_used_ratio` = `quota_used` / `quota_size`
- `arena_used_ratio` = `arena_used` / `arena_size`

**тип возвращаемого значения** таблица

**Пример:**

```
tarantool> box.slab.info()
---
- items_size: 228128
  items_used_ratio: 1.8%
  quota_size: 1073741824
  quota_used_ratio: 0.8%
  arena_used_ratio: 43.2%
  items_used: 4208
  quota_used: 8388608
  arena_size: 2325176
  arena_used: 1003632
...

tarantool> box.slab.info().arena_used
---
- 1003632
...
```

### box.slab.stats()

```
box.slab.stats()
```

Отображение подробного отчета об использовании памяти (в байтах) для распределения slab. Отчет разбивается на группы по *размеру элементов данных*, а также по *размеру slab'а* (64 байта, 136 байтов и т.д.). Отчет включает в себя информацию о памяти, выделенной на хранение и кортежей, и индексов.

**возвращает**

- `mem_free` – это выделенная, но не используемая в данный момент память;
- `mem_used` – это память, используемая для хранения элементов данных (кортежей и индексов);
- `item_count` – это количество хранимых элементов;
- `item_size` – это размер каждого элемента данных;
- `slab_count` – это количество выделенных slab'ов;
- `slab_size` – это размер каждого выделенного slab'а.

**тип возвращаемого значения** таблица

**Пример:**

Ниже представлен пример отчета для первой группы:

```
tarantool> box.slab.stats()[1]
---
- mem_free: 16232
  mem_used: 48
  item_count: 2
  item_size: 24
  slab_count: 1
  slab_size: 16384
...
```

В отчете показано, что есть два элемента данных (`item_count` = 2), которые хранятся в одном (`slab_count` = 1) 24-байтовом slab'е (`item_size` = 24), поэтому объем используемой памяти `mem_used` = 2 * 24 = 48 байтов. Кроме того, размер slab'а `slab_size` составляет 16384 байта, из которых 16384 - 48 = 16232 байта свободны (`mem_free`).

В полном отчете будет статистика по использованию памяти во всех группах:

```
tarantool> box.slab.stats()
---
- - mem_free: 16232
    mem_used: 48
    item_count: 2
    item_size: 24
    slab_count: 1
    slab_size: 16384
  - mem_free: 15720
    mem_used: 560
    item_count: 14
    item_size: 40
    slab_count: 1
    slab_size: 16384
  <...>
  - mem_free: 32472
    mem_used: 192
    item_count: 1
    item_size: 192
    slab_count: 1
    slab_size: 32768
  - mem_free: 1097624
    mem_used: 999424
    item_count: 61
    item_size: 16384
    slab_count: 1
    slab_size: 2097152
  ...
```

The total `mem_used` for all groups in this report equals `arena_used` in *box.slab.info()* report.

### Вложенный модуль box.stat

Вложенный модуль `box.stat` предоставляет доступ к статистике Tarantool по запросам и использованию сети.

Below is a list of all `box.stat` functions.

| Name | Use |
|---|---|
| *box.stat()* | Show request statistics |
| *box.stat.net()* | Show network activity |
| *box.stat.vinyl()* | Show vinyl-storage-engine activity |
| *box.stat.reset()* | Reset the statistics |

### box.stat()

`box.stat()`
Shows the total number of requests since startup and the average number of requests per second, broken down by request type.

**Return** in the tables that `box.stat()` returns:

- `total`: total number of requests processed per second since the server started
- `rps`: average number of requests per second in the last 5 seconds.

`ERROR` is the count of requests that resulted in an error.

**Example:**

```
tarantool> box.stat() -- return 15 tables
---
- DELETE:
    total: 0
    rps: 0
  COMMIT:
    total: 0
    rps: 0
  SELECT:
    total: 12
    rps: 0
  ROLLBACK:
    total: 0
    rps: 0
  INSERT:
    total: 6
    rps: 0
  EVAL:
    total: 0
    rps: 0
  ERROR:
    total: 0
    rps: 0
  CALL:
    total: 0
    rps: 0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  BEGIN:
    total: 0
    rps: 0
  PREPARE:
    total: 0
    rps: 0
  REPLACE:
    total: 0
    rps: 0
  UPSERT:
    total: 0
    rps: 0
  AUTH:
    total: 0
    rps: 0
  EXECUTE:
    total: 0
    rps: 0
  UPDATE:
    total: 2
    rps: 0
...

tarantool> box.stat().DELETE -- total + requests per second from one table
---
- total: 0
  rps: 0
...
```

### box.stat.net()

`box.stat.net()`

Shows network activity: the number of bytes sent and received, the number of connections, streams, and requests (current, average, and total).

**Return** in the tables that `box.stat.net()` returns:

- `SENT.rps` and `RECEIVED.rps` – average number of bytes sent/received per second in the last 5 seconds
- `SENT.total` and `RECEIVED.total` – total number of bytes sent/received since the server started
- `CONNECTIONS.current` – number of open connections
- `CONNECTIONS.rps` – average number of connections opened per second in the last 5 seconds
- `CONNECTIONS.total` – total number of connections opened since the server started
- `REQUESTS.current` – number of requests in progress, which can be limited by *box.cfg.net_msg_max*
- `REQUESTS.rps` – average number of requests processed per second in the last 5 seconds

- `REQUESTS.total` – total number of requests processed since the server started
- `REQUESTS_IN_PROGRESS.current` – number of requests being currently processed by the *TX thread*
- `REQUESTS_IN_PROGRESS.rps` – average number of requests processed by the TX thread per second in the last 5 seconds
- `REQUESTS_IN_PROGRESS.total` – total number of requests processed by the TX thread since the server started
- `STREAMS.current` – number of active *streams*
- `STREAMS.rps` – average number of streams opened per second in the last 5 seconds
- `STREAMS.total` – total number of streams opened since the server started
- `REQUESTS_IN_STREAM_QUEUE.current` – number of requests waiting in stream queues
- `REQUESTS_IN_STREAM_QUEUE.rps` – average number of requests in stream queues per second in the last 5 seconds
- `REQUESTS_IN_STREAM_QUEUE.total` – total number of requests placed in stream queues since the server started

**Example:**

```
tarantool> box.stat.net() -- 5 tables
---
- CONNECTIONS:
    current: 1
    rps: 0
    total: 1
  REQUESTS:
    current: 0
    rps: 0
    total: 8
  REQUESTS_IN_PROGRESS:
    current: 0
    rps: 0
    total: 7
  SENT:
    total: 19579
    rps: 0
  REQUESTS_IN_STREAM_QUEUE:
    current: 0
    rps: 0
    total: 0
  STREAMS:
    current: 0
    rps: 0
    total: 0
  RECEIVED:
    total: 197
    rps
...
```

net.thread()

Shows network activity per *network thread*: the number of bytes sent and received, the number of connections, streams, and requests (current, average, and total).

When called with an index (box.stat.net.thread[1]), shows network statistics for a single network thread.

**Return** Same network activity metrics as *box.stat.net()* for each network thread

**Example:**

```
tarantool> box.stat.net.thread() -- iproto_threads = 2
- - CONNECTIONS:
      current: 0
      rps: 0
      total: 0
    REQUESTS:
      current: 0
      rps: 0
      total: 0
    REQUESTS_IN_PROGRESS:
      current: 0
      rps: 0
      total: 0
    SENT:
      total: 0
      rps: 0
    REQUESTS_IN_STREAM_QUEUE:
      current: 0
      rps: 0
      total: 0
    STREAMS:
      current: 0
      rps: 0
      total: 0
    RECEIVED:
      total: 0
      rps: 0
  - CONNECTIONS:
      current: 1
      rps: 0
      total: 1
    REQUESTS:
      current: 0
      rps: 0
      total: 8
    REQUESTS_IN_PROGRESS:
      current: 0
      rps: 0
      total: 7
    SENT:
      total: 19579
      rps: 0
    REQUESTS_IN_STREAM_QUEUE:
      current: 0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      rps: 0
      total: 0
    STREAMS:
      current: 0
      rps: 0
      total: 0
    RECEIVED:
      total: 197
      rps: 0
...
```

```
tarantool> box.stat.net.thread[1] -- first network thread
- - CONNECTIONS:
      current: 1
      rps: 0
      total: 1
    REQUESTS:
      current: 0
      rps: 0
      total: 8
    REQUESTS_IN_PROGRESS:
      current: 0
      rps: 0
      total: 7
    SENT:
      total: 19579
      rps: 0
    REQUESTS_IN_STREAM_QUEUE:
      current: 0
      rps: 0
      total: 0
    STREAMS:
      current: 0
      rps: 0
      total: 0
    RECEIVED:
      total: 197
      rps: 0
...
```

### box.stat.vinyl()

box.stat.vinyl()
Shows vinyl-storage-engine activity, for example `box.stat.vinyl().tx` has the number of commits and rollbacks.

**Example:**

```
tarantool> box.stat.vinyl().tx.commit -- one item of the vinyl table
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- 1047632
...
```

### box.stat.vinyl().regulator

The vinyl regulator decides when to take or delay actions for disk IO, grouping activity in batches so that it is consistent and efficient. The regulator is invoked by the vinyl scheduler, once per second, and updates related variables whenever it is invoked.

- `box.stat.vinyl().regulator.dump_bandwidth` is the estimated average rate at which dumps are done. Initially this will appear as 10485760 (10 megabytes per second). Only significant dumps (larger than one megabyte) are used for estimating.
- `box.stat.vinyl().regulator.dump_watermark` is the point when dumping must occur. The value is slightly smaller than the amount of memory that is allocated for vinyl trees, which is the *vinyl_memory* parameter.
- `box.stat.vinyl().regulator.write_rate` is the actual average rate at which recent writes to disk are done. Averaging is done over a 5-second time window, so if there has been no activity for 5 seconds then `regulator.write_rate = 0`. The `write_rate` may be slowed when a dump is in progress or when the user has set *snap_io_rate_limit*.
- `box.stat.vinyl().regulator.rate_limit` is the write rate limit, in bytes per second, imposed on transactions by the regulator based on the observed dump/compaction performance.
- `box.stat.vinyl().regulator.blocked_writers` is the number of fibers currently blocked waiting for vinyl *L0 memory* quota.

### box.stat.vinyl().disk

Since vinyl is an on-disk storage engine (unlike memtx which is an in-memory storage engine), it can handle large databases – but if a database is larger than the amount of memory that is allocated for vinyl, then there will be more disk activity.

- `box.stat.vinyl().disk.data` and `box.stat.vinyl().disk.index` are the amount of data that has gone into files in a subdirectory of *vinyl_dir*, with names like `{lsn}.run` and `{lsn}.index`. The size of the run will be related to the output of `scheduler.dump_*`.
- `box.stat.vinyl().disk.data_compacted` Sum size of data stored at the last LSM tree level, in bytes, without taking disk compression into account. It can be thought of as the size of disk space that the user data would occupy if there were no compression, indexing, or space increase caused by the LSM tree design.

### box.stat.vinyl().memory

Although the vinyl storage engine is not «in-memory», Tarantool does need to have memory for write buffers and for caches:

- `box.stat.vinyl().memory.tuple_cache` is the number of bytes that are being used for tuples (data).
- `box.stat.vinyl().memory.tx` is transactional memory. This will usually be 0.
- `box.stat.vinyl().memory.level0` is the «level0» memory area, sometimes abbreviated «L0», which is the area that vinyl can use for in-memory storage of an LSM tree.

Therefore we can say that «L0 is becoming full» when the amount in `memory.level0` is close to the maximum, which is *regulator.dump_watermark*. We can expect that «L0 = 0» immediately after a dump. `box.stat.vinyl().memory.page_index` and `box.stat.vinyl().memory.bloom_filter` have the current amount being used for index-related structures. The size is a function of the number and size of keys, plus *vinyl_page_size*, plus *vinyl_bloom_fpr*. This is not a count of bloom filter «hits» (the number of reads that could be avoided because the bloom filter predicts their presence in a run file) – that statistic can be found with *index_object:stat()*.

### box.stat.vinyl().tx

This is about requests that affect transactional activity («tx» is used here as an abbreviation for «transaction»):

- `box.stat.vinyl().tx.conflict` counts conflicts that caused a transaction to roll back.
- `box.stat.vinyl().tx.commit` is the count of commits (successful transaction ends). It includes implicit commits, for example any insert causes a commit unless it is within a begin-end block.
- `box.stat.vinyl().tx.rollback` is the count of rollbacks (unsuccessful transaction ends). This is not merely a count of explicit *box.rollback()* requests – it includes requests that ended in errors. For example, after an attempted insert request that causes a «Duplicate key exists in unique index» error, `tx.rollback` is incremented.
- `box.stat.vinyl().tx.statements` will usually be 0.
- `box.stat.vinyl().tx.transactions` is the number of transactions that are currently running.
- `box.stat.vinyl().tx.gap_locks` is the number of gap locks that are outstanding during execution of a request. For a low-level description of Tarantool's implementation of gap locking, see Gap locks in Vinyl transaction manager.
- `box.stat.vinyl().tx.read_views` shows whether a transaction has entered a read-only state to avoid conflict temporarily. This will usually be 0.

### box.stat.vinyl().scheduler

This primarily has counters related to tasks that the scheduler has arranged for dumping or compaction: (most of these items are reset to 0 when the server restarts or when *box.stat.reset()* occurs):

- `box.stat.vinyl().scheduler.compaction_*` is the amount of data from recent changes that has been *compacted*. This is divided into `scheduler.compaction_input` (the amount that is being compacted), `scheduler.compaction_queue` (the amount that is waiting to be compacted), `scheduler.compaction_time` (total time spent by all worker threads performing compaction, in seconds), and `scheduler.compaction_output` (the amount that has been compacted, which is presumably smaller than `scheduler.compaction_input`).
- `box.stat.vinyl().scheduler.tasks_*` is about dump/compaction tasks, in three categories, `scheduler.tasks_inprogress` (currently running), `scheduler.tasks_completed` (successfully completed) `scheduler.tasks_failed` (aborted due to errors).
- `box.stat.vinyl().scheduler.dump_*` has the amount of data from recent changes that has been dumped, including `dump_time` (total time spent by all worker threads performing dumps, in seconds), and `dump_count` (the count of completed dumps), `dump_input` and `dump_output`.

  A «dump» is explained in section *Storing data with vinyl*:

Sooner or later the number of elements in an LSM tree exceeds the L0 size and that is when L0 gets written to a file on disk (called a „run“) and then cleared for storing new elements. This operation is called a „dump“.

Thus it can be predicted that a dump will occur if the size of L0 (which is *memory.level0*) is approaching the maximum (which is *regulator.dump_watermark*) and a dump is not already in progress. In fact Tarantool will try to arrange a dump before this hard limit is reached.

A dump will also occur during a *snapshot* operation.

### box.stat.reset()

`box.stat.reset()`
Resets the statistics of `box.stat()`, `box.stat.net()`, `box.stat.vinyl()`, and *box.space.index*.

### Вложенный модуль box.tuple

Вложенный модуль `box.tuple` предоставляет доступ только для чтения к пользовательским данным типа кортеж `tuple`. С его помощью для отдельного *кортежа* можно сделать следующее: выборочно искать содержимое поля, получать информацию о размере, проводить итерацию по всем полям и выполнять преобразование в Lua-таблицу.

Ниже приведен перечень всех функций модуля `box.tuple`.

| Имя | Использование |
|---|---|
| *box.tuple.new()* | Создание кортежа |
| *box.tuple.is()* | Проверка, является ли переданный объект кортежем |
| *#tuple_object* | Подсчет полей кортежа |
| *box.tuple.bsize()* | Подсчет байтов в кортеже |
| *tuple_object[field-number]* | Получение поля кортежа по номеру |
| *tuple_object[field-name]* | Получение поля кортежа по имени |
| *tuple_object[field-path]* | Получение полей кортежа или компонентов по пути |
| *tuple_object:find(), tuple_object:findall()* | Получение номера первого поля, совпадающего с искомым значением |
| *tuple_object:next()* | Получение значения следующего поля из кортежа |
| *tuple_object:pairs(), tuple_object:ipairs()* | Подготовка к итерации |
| *tuple_object:totable()* | Получение полей кортежа в виде таблицы |
| *tuple_object:tomap()* | Получение полей кортежа в виде таблицы, а также пар ключ-значение |
| *tuple_object:transform()* | Удаление (и замена) полей кортежа |
| *tuple_object:unpack()* | Получение полей кортежа |
| *tuple_object:update()* | Обновление кортежа |
| *tuple_object:upsert()* | Обновление кортежа, игнорируя ошибки |

### box.tuple.new()

box.tuple.new(*value*)

Создание нового кортежа либо из скаляра, либо из Lua-таблицы. Возможен и вариант получения новых кортежей из запросов *select* или *insert*. или *replace*, или *update* Tarantool, которые можно рассматривать в качестве операторов, косвенно выполняющих операцию создания `new()`.

**Параметры**

- `value` (`lua-value`) – значение, которое станет содержимым кортежа.

**возвращает** новый кортеж

**тип возвращаемого значения** кортеж

В следующем примере `x` будет представлять собой новый объект таблицы, который содержит один кортеж, а `t` будет представлять собой объект кортежа. Если ввести команду `t`, будет получен весь кортеж `t`.

**Пример:**

```
tarantool> x = box.space.tester:insert{
         >   33,
         >   tonumber('1'),
         >   tonumber64('2')
         > }:totable()
---
...
tarantool> t = box.tuple.new{'abc', 'def', 'ghi', 'abc'}
---
...
tarantool> t
---
- ['abc', 'def', 'ghi', 'abc']
...
```

### box.tuple.is()

box.tuple.is(*object*)

Since versions *2.2.3*, *2.3.2*, and *2.4.1*. A function to check whether a given object is a tuple cdata object. Never raises nor returns an error.

**возвращает** true или false

**возвращаемое значение** boolean (логический)

### #tuple_object

```
object tuple_object
```

#<tuple_object>

Оператор # на языке Lua означает «вернуть количество компонентов». Таким образом, если t представляет собой кортеж, то #t вернет количество полей.

**тип возвращаемого значения** число

В следующем примере создается кортеж под названием t, а затем возвращается количество полей в кортеже t.

```
tarantool> t = box.tuple.new{'Fld#1', 'Fld#2', 'Fld#3', 'Fld#4'}
---
...
tarantool> #t
---
- 4
...
```

### box.tuple.bsize()

```
object tuple_object
```

tuple_object:bsize()

Если t – это экземпляр кортежа, то t:bsize() вернет количество байтов в кортеже. Как для движка базы данных memtx, так и для движка vinyl максимальное количество, используемое по умолчанию, составляет один мегабайт (*memtx_max_tuple_size* или *vinyl_max_tuple_size*). В каждом поле есть один или более байтов «длины», которые предваряют само содержимое поля, поэтому bsize() вернет значение, которое незначительно больше, чем сумма длин всего содержимого.

Значение не содержит размер кортежа «struct tuple» (чтобы узнать текущий размер данной структуры, посмотрите файл tuple.h в исходном коде Tarantool).

**возвращает** количество байтов

**тип возвращаемого значения** число

В следующем примере создается кортеж с именем t, в котором три поля, и для каждого поля один байт занимает хранение длины, и три байта занимает хранение содержимого, кроме того, один бит используется на ресурсы, поэтому bsize() вернет 3*(1+3)+1. Такой же размер строки вернула бы функция *msgpack.encode({„aaa“,“bbb“,“ccc“})*.

```
tarantool> t = box.tuple.new{'aaa', 'bbb', 'ccc'}
---
...
tarantool> t:bsize()
---
- 13
...
```

### tuple_object[field-number]

```
object tuple_object
```

<tuple_object>(*field-number*)

Если `t` – это экземпляр кортежа, то `t[номер-поля]` вернет поле под номером номер-поля в кортеже. Первое поле – это `t[1]`.

**возвращает** значение поля.

**тип возвращаемого значения** Lua-значение

В следующем примере создается кортеж под названием `t`, а затем возвращается второе поле в кортеже `t`.

```
tarantool> t = box.tuple.new{'Fld#1', 'Fld#2', 'Fld#3', 'Fld#4'}
---
...
tarantool> t[2]
---
- Fld#2
...
```

### tuple_object[field-name]

```
object tuple_object
```

<tuple_object>(*field-name*)

Если `t` – это экземпляр кортежа, то `t['field-name']` вернет поле под названием `field-name` в кортеже. У полей есть имена, если кортеж был получен из спейса с определенным *формат*ом. `t[lua-variable-name]` сделает то же самое, если `lua-variable-name` содержит `'field-name'`.

Есть вариация, которую Lua manual называет «синтаксический сахар»: можно использовать `t.field-name` вместо `t['field-name']`.

**возвращает** значение поля

**тип возвращаемого значения** Lua-значение

В следующем примере кортеж под названием `t` возвращается после операции замены, а затем возвращается второе поле с именем „field2“ в кортеже `t`.

```
tarantool> format = {}
---
...
tarantool> format[1] = {name = 'field1', type = 'unsigned'}
---
...
tarantool> format[2] = {name = 'field2', type = 'string'}
---
...
tarantool> s = box.schema.space.create('test', {format = format})
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> pk = s:create_index('pk')
---
...
tarantool> t = s:replace{1, 'Я'}
---
...
tarantool> t['field2']
---
- Я
...
```

### tuple_object[field-path]

```
object tuple_object
```

<tuple_object>(*field-path*)

Если `t` – это экземпляр кортежа, то `t['path']` вернет поле или ряд полей, которые находятся в `path`. Параметр `path` должен представлять собой правильную JSON-спецификацию. `path` может содержать имена полей, если кортеж был получен из спейса с заданным *форматом*.

Во избежание неоднозначности Tarantool сначала пытается интерпретировать запрос как *tuple_object[field-number]* или *tuple_object[field-name]*. И только в том случае, если это не удается, Tarantool пытается интерпретировать запрос как `tuple_object[field-path]`.

Путь path должен представлять собой правильную JSON-спецификацию, но в начале может стоять „.“. Символ „.“ означает, что путь выступает в качестве суффикса для кортежа.

При указании пути Tarantool воспользуется им для поиска по телу кортежа и вернет только тот компонент кортежа, который действительно необходим.

In the following example, a tuple named `t` is returned from `replace` and then only the relevant part (in this case, matching a name) of a relevant field is returned. Namely: the second field, its third item, the value following „key=“.

```
tarantool> format = {}
---
...
tarantool> format[1] = {name = 'field1', type = 'unsigned'}
---
...
tarantool> format[2] = {name = 'field2', type = 'array'}
---
...
tarantool> s = box.schema.space.create('test', {format = format})
---
...
tarantool> pk = s:create_index('pk')
---
...
tarantool> field2_value = {1, "ABC", {key="Hello", value="world"}}
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> t = s:replace{1, field2_value}
---
...
tarantool> t["[2][3]['key']"]
---
- Hello
...
```

### tuple_object:find(), tuple_object:findall()

```
object tuple_object
```

tuple_object:find([*field-number*], *search-value*)
tuple_object:findall([*field-number*], *search-value*)

Если `t` – это экземпляр кортежа, то `t:find(search-value)` вернет номер первого поля в `t`, которое совпадает с искомым значением, а `t:findall(search-value [, search-value ...])` вернет номера всех колей в `t`, которые совпадают с искомым значением. Можно дополнительно добавить числовой аргумент `field-number` перед search-value, чтобы задать условие "начинать поиск с номера поля `field-number`."

**возвращает** номер поля в кортеже.

**тип возвращаемого значения** число

В следующем примере создается кортеж с именем `t`, а затем: возвращается номер первого поля в `t`, которое совпадает с „a“, затем возвращаются номера всех полей в `t`, которые совпадают с „a“, затем возвращаются номера всех полей в t, которые совпадают с „a“, и находятся на втором месте или далее.

```
tarantool> t = box.tuple.new{'a', 'b', 'c', 'a'}
---
...
tarantool> t:find('a')
---
- 1
...
tarantool> t:findall('a')
---
- 1
- 4
...
tarantool> t:findall(2, 'a')
---
- 4
...
```

### tuple_object:next()

```
object tuple_object
```

tuple_object:next(*tuple*[, *pos*])

Аналог функции Lua `next()`, но для кортежа. При вызове без аргументов, `tuple:next()` возвращает первое поле из кортежа. В противном случае возвращается поле рядом с указанной позицией.

Однако `tuple:next()` не очень эффективен, и лучше использовать *tuple:pairs()/ipairs()*.

**возвращает** номер и значение поля

**тип возвращаемого значения** число (number) и тип поля

```
tarantool> tuple = box.tuple.new({5, 4, 3, 2, 0})
---
...

tarantool> tuple:next()
---
- 1
- 5
...

tarantool> tuple:next(1)
---
- 2
- 4
...

tarantool> ctx, field = tuple:next()
---
...

tarantool> while field do
         > print(field)
         > ctx, field = tuple:next(ctx)
         > end
5
4
3
2
0
---
...
```

### tuple_object:pairs(), tuple_object:ipairs()

object tuple_object

tuple_object:pairs()
tuple_object:ipairs()

В языке Lua метод lua-table-value:pairs() возвращает: `функция`, `значение-Lua-таблицы`, `nil`. В Tarantool метод расширен так, что `tuple-value:pairs()` возвращает: `функция`, `значение-кортежа`, `nil`, – что используется для Lua-итераторов, поскольку они обходят компоненты значения до тех пор, пока не достигнут маркера.

`tuple_object:ipairs()` – это то же самое, что и `pairs()`, потому что поля кортежей всегда являются натуральными числами.

**возвращает** функция, значение кортежа, nil

**тип возвращаемого значения** функция, Lua-значение, nil

В следующем примере создается кортеж под названием `t`, а затем все его поля выбираются с помощью Lua-цикла for.

```
tarantool> t = box.tuple.new{'Fld#1', 'Fld#2', 'Fld#3', 'Fld#4', 'Fld#5'}
---
...
tarantool> tmp = ''
---
...
tarantool> for k, v in t:pairs() do
         >   tmp = tmp .. v
         > end
---
...
tarantool> tmp
---
- Fld#1Fld#2Fld#3Fld#4Fld#5
...
```

### tuple_object:totable()

object tuple_object

tuple_object:totable(⌊*start-field-number*⌊, *end-field-number*⌋⌋)

Если `t` – это экземпляр кортежа, то `t:totable()` вернет все поля, `t:totable(1)` вернет все поля, начиная с поля №1, `t:totable(1,5)` вернет все поля между полем №1 и полем №5.

Рекомендуется использовать `t:totable()`, а не `t:unpack()`.

**возвращает** поле или поля из кортежа

**тип возвращаемого значения** Lua-таблица

В следующем примере создается кортеж под названием `t`, а затем делается выборка всех полей, возвращается результат.

```
tarantool> t = box.tuple.new{'Fld#1', 'Fld#2', 'Fld#3', 'Fld#4', 'Fld#5'}
---
...
tarantool> t:totable()
---
- ['Fld#1', 'Fld#2', 'Fld#3', 'Fld#4', 'Fld#5']
...
```

### tuple_object:tomap()

```
object tuple_object
```

tuple_object:tomap([*options*])

В Lua-таблице могут быть индексированные значения, которые также называются пары ключ-значение. Например, здесь:

```
a = {}; a['field1'] = 10; a['field2'] = 20
```

a – это таблица с «field1: 10» и «field2: 20».

Функция *tuple_ object:totable()* вернет только таблицу со значениями. А функция tuple_object:tomap() вернет таблицу не только со значениями, но и с парами ключ-значение.

Это сработает только в том случае, если кортеж приходит из спейса, который был форматирован посредством *оператора формата*.

**Параметры**

- options (*table*) – единственный доступный параметр – names_only. Если names_only принимает значение false или не указан (по умолчанию), то все поля появятся дважды: сначала с числовыми заголовками, а затем с именными заголовками. Если же names_only = true, то все поля будут выведены один раз с именными заголовками.

**возвращает** пары номер-поля:значение и пары ключ:значение из кортежа

**тип возвращаемого значения** Lua-таблица

В следующем примере возвращается кортеж с именем t1 из спейса после форматирования, затем таблицы с именами t1map и t1map2 создаются из t1.

```
format = {{'field1', 'unsigned'}, {'field2', 'unsigned'}}
s = box.schema.space.create('test', {format = format})
s:create_index('pk',{parts={1,'unsigned',2,'unsigned'}})
t1 = s:insert{10, 20}
t1map = t1:tomap()
t1map_names_only = t1:tomap({names_only=true})
```

t1map будет содержать «1: 10», «2: 20», «field1: 10», «field2: 20».

t1map_names_only будет содержать «field1: 10» и «field2: 20».

### tuple_object:transform()

```
object tuple_object
```

tuple_object:transform(*start-field-number*, *fields-to-remove*[, *field-value*, ...])

Если `t` – это экземпляр кортежа, то `t:transform(start-field-number,fields-to-remove)` вернет кортеж, где начиная с поля `start-field-number`, удаляется количество полей (`fields-to-remove`). Дополнительно можно добавить аргументы после `fields-to-remove`, чтобы указать новые значения на замену удаленных.

Если первоначальный кортеж приходит из спейса, который был форматирован посредством *оператора формата*, форматирование возвращаемого кортежа не сохранится.

**Параметры**

- `start-field-number` (`integer`) – начиная с 1, может быть отрицательным
- `fields-to-remove` (`integer`) –
- `field-value(s)` (`lua-value`) –

**возвращает** tuple

**тип возвращаемого значения** tuple

В следующем примере создается кортеж под названием `t`, а затем начиная со второго поля, удаляются два поля, а одно новое поле добавляется, затем возвращается результат.

```
tarantool> t = box.tuple.new{'Fld#1', 'Fld#2', 'Fld#3', 'Fld#4', 'Fld#5'}
---
...
tarantool> t:transform(2, 2, 'x')
---
- ['Fld#1', 'x', 'Fld#4', 'Fld#5']
...
```

### tuple_object:unpack()

```
object tuple_object
```

tuple_object:unpack([*start-field-number*[, *end-field-number*]])

Если `t` – это экземпляр кортежа, то `t:unpack()` вернет все поля, `t:unpack(1)` вернет все поля, начиная с поля №1, `t:unpack(1,5)` вернет все поля между полем №1 и полем №5.

**возвращает** поле или поля из кортежа.

**тип возвращаемого значения** Lua-значение(я)

В следующем примере создается кортеж под названием `t`, а затем делается выборка всех полей, возвращается результат.

```
tarantool> t = box.tuple.new{'Fld#1', 'Fld#2', 'Fld#3', 'Fld#4', 'Fld#5'}
---
...
tarantool> t:unpack()
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- Fld#1
- Fld#2
- Fld#3
- Fld#4
- Fld#5
...
```

### tuple_object:update()

```
object tuple_object
```

tuple_object:update(*{{operator, field_no, value}, ...}*)

Обновление кортежа.

Эта функция обновляет кортеж, который находится не в спейсе. Ср. функцию box.space.*space-name*:update(*key*, {{*format*, *field_no*, *value*}, ...}), которая обновляет кортеж в спейсе.

Более подробную информацию см. в описании operator, field_no и value в разделе *box.space.space-name:update{key, format, {field_number, value}...)*.

Если первоначальный кортеж приходит из спейса, который был форматирован посредством *оператора формата*, форматирование возвращаемого кортежа сохранится.

**Параметры**

- operator (*string*) – тип операции, представленный строкой (например, „=“ означает „присвоить новое значение“)
- field_no (number) – к какому полю применяется операция. Номер поля может быть отрицательным, что означает, что позиция рассчитывается с конца кортежа. (#кортеж + отрицательный номер поля + 1)
- value (lua_value) – какое значение применяется

**возвращает** новый кортеж

**тип возвращаемого значения** кортеж

В следующем примере создается кортеж под названием t, а затем второе поле обновляется до равного „B“.

```
tarantool> t = box.tuple.new{'Fld#1', 'Fld#2', 'Fld#3', 'Fld#4', 'Fld#5'}
---
...
tarantool> t:update({{'=', 2, 'B'}})
---
- ['Fld#1', 'B', 'Fld#3', 'Fld#4', 'Fld#5']
...
```

Начиная с версии Tarantool 2.3 кортеж можно обновить с помощью *JSON-путей*.

### tuple_object:upsert()

```
object tuple_object
```

tuple_object:upsert(*{{operator, field_no, value}, ...}*)

То же самое, что и `tuple_object:update()`, но игнорирует ошибки. В случае ошибки кортеж остаётся нетронутым, но выводится сообщение об ошибке. Игнорируются только клиентские ошибки, такие как «плохой тип поля» или «неправильный индекс/имя» поля. Системные ошибки, такие как OOM, не игнорируются и поднимаются так же, как и при обычном `update()`. Обратите внимание, что игнорируются только некорректные операции. Все корректные операции применяются.

**Параметры**

- `operator` (*string*) – тип операции, представленный строкой (например, „=“ означает „присвоить новое значение“)
- `field_no` (`number`) – к какому полю применяется операция. Номер поля может быть отрицательным, что означает, что позиция рассчитывается с конца кортежа. (#кортеж + отрицательный номер поля + 1)
- `value` (`lua_value`) – значение, которое применяется

**возвращает** новый кортеж

**тип возвращаемого значения** tuple

В следующем примере одна операция применяется, а другая – нет.

```
tarantool> t = box.tuple.new({1, 2, 3})
tarantool> t2 = t:upsert({{'=', 5, 100}})
UPSERT operation failed:
ER_NO_SUCH_FIELD_NO: Field 5 was not found in the tuple
---
...

tarantool> t
---
- [1, 2, 3]
...

tarantool> t2
---
- [1, 2, 3]
...

tarantool> t2 = t:upsert({{'=', 5, 100}, {'+', 1, 3}})
UPSERT operation failed:
ER_NO_SUCH_FIELD_NO: Field 5 was not found in the tuple
---
...

tarantool> t
---
- [1, 2, 3]
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> t2
---
- [4, 2, 3]
...
```

### Как преобразовать кортежи в Lua-таблицы и обратно

Представленная ниже функция проиллюстрирует, как можно преобразовать кортежи в Lua-таблицы и списки скаляров и обратно:

```
tuple = box.tuple.new({scalar1, scalar2, ... scalar_n}) -- скаляры в кортеж
lua_table = {tuple:unpack()}                            -- кортеж в Lua-таблицу
lua_table = tuple:totable()                             -- кортеж в Lua-таблицу
scalar1, scalar2, ... scalar_n = tuple:unpack()         -- кортеж в скаляры
tuple = box.tuple.new(lua_table)                        -- Lua-таблицу в кортеж
```

Затем она найдет поле, которое содержит значение „b“, удалит это поле из кортежа и отобразит количество байтов, оставшихся в кортеже. Данная функция использует следующие функции `box.tuple` Tarantool: `new()`, `unpack()`, `find()`, `transform()`, `bsize()`.

```
function example()
  local tuple1, tuple2, lua_table_1, scalar1, scalar2, scalar3, field_number
  local luatable1 = {}
  tuple1 = box.tuple.new({'a', 'b', 'c'})
  luatable1 = tuple1:totable()
  scalar1, scalar2, scalar3 = tuple1:unpack()
  tuple2 = box.tuple.new(luatable1[1],luatable1[2],luatable1[3])
  field_number = tuple2:find('b')
  tuple2 = tuple2:transform(field_number, 1)
  return 'tuple2 = ' , tuple2 , ' # of bytes = ' , tuple2:bsize()
end
```

... А вот что происходит, когда вызывается функция:

```
tarantool> example()
---
- tuple2 =
- ['a', 'c']
- ' # of bytes = '
- 5
...
```

### Управление экземплярами

For general information and examples, see section *Transactions*.

Соблюдайте следующие правила в работе с транзакциями:

---

**Правило #1**

Запросы в транзакции должны отправляться на сервер в виде единого блока. Недостаточно просто размещать их между началом транзакции и коммитом или откатом. Чтобы убедиться, что они отправляются в виде единого блока: поместите их в функцию, поместите их на одну строку или используйте символы-разделители, чтобы многостроковые запросы обрабатывались совместно.

---

**Правило #2**

Все операции с базой данных в рамках транзакции должны работать с одним движком баз данных. Небезопасно в рамках одной транзакции получать доступ к наборам кортежей, которые определяются по `{engine='vinyl'}`, а также к наборам кортежей, которые определяются по `{engine='memtx'}`.

---

**Правило #3**

Requests which cause changes to the data definition – create, alter, drop, truncate – are only allowed with Tarantool version 2.1 or later. Data-definition requests which change an index or change a format, such as *space_object:create_index()* and *space_object:format()*, are not allowed inside transactions except as the first request after `box.begin()`.

---

Ниже приведен перечень всех функций для управления транзакциями.

| Имя | Использование |
|---|---|
| *box.begin()* | Начало транзакции |
| *box.commit()* | Окончание транзакции и сохранение всех изменений |
| *box.rollback()* | Окончание транзакции и отмена всех изменений |
| *box.savepoint()* | Получение дескриптора точки сохранения |
| *box.rollback_to_savepoint()* | Запрещение окончания транзакции и отмена всех изменений, сделанных после точки сохранения |
| *box.atomic()* | Выполнение функции как транзакции |
| *box.on_commit()* | Определение триггера, активируемого по `box.commit` |
| *box.on_rollback()* | Определение триггера, активируемого по `box.rollback` |
| *box.is_in_txn()* | Обозначение наличия активной транзакции |

### box.begin()

`box.begin`([*opts*])

Начало транзакции. Отключение неявной передачи управления до окончания транзакции. Сигнал о записи в *журнал упреждающей записи* будет задержан до окончания транзакции. Фактически файбер, который выполняет функцию `box.begin()`, начинает «активную транзакцию со множеством запросов» с блокировкой всех остальных файберов.

**Параметры**

- `opts` (*table*) – (optional) transaction options:

– `txn_isolation` – the *transaction isolation level*

– `timeout` – a timeout (in seconds), after which the transaction is rolled back

**Возможные ошибки:**

- ошибка, если такая операция не допускается, потому что уже есть активная транзакция.
- ошибка, если по какой-либо причине нельзя выделить память.
- error and abort the transaction if the timeout is exceeded.

**Example**

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }

-- Begin and commit the transaction explicitly --
box.begin()
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.space.bands:replace { 1, 'Pink Floyd', 1965 }
box.commit()

-- Begin the transaction with the specified isolation level --
box.begin({ txn_isolation = 'read-committed' })
box.space.bands:insert { 5, 'The Rolling Stones', 1962 }
box.space.bands:replace { 1, 'The Doors', 1965 }
box.commit()
```

### box.commit()

`box.commit()`

Окончание транзакции и применение результатов всех операций по изменению данных.

**Возможные ошибки:**

- ошибка и прерывание транзакции в случае конфликта.
- ошибка, если операция не может выполнить запись на диск.
- ошибка, если по какой-либо причине нельзя выделить память.

**Example**

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }

-- Begin and commit the transaction explicitly --
box.begin()
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.space.bands:replace { 1, 'Pink Floyd', 1965 }
box.commit()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Begin the transaction with the specified isolation level --
box.begin({ txn_isolation = 'read-committed' })
box.space.bands:insert { 5, 'The Rolling Stones', 1962 }
box.space.bands:replace { 1, 'The Doors', 1965 }
box.commit()
```

### box.rollback()

box.rollback()

Окончание транзакции, но отмена результатов всех операций по изменению данных. Явный вызов функций не из модуля `box.space`, которые всегда передают управление, например *fiber.sleep()* или *fiber.yield()*, приведет к тому же результату.

**Example**

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }

-- Rollback the transaction --
box.begin()
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.space.bands:replace { 1, 'Pink Floyd', 1965 }
box.rollback()
```

### box.savepoint()

box.savepoint()

Выдача дескриптора точки сохранения (тип = таблица), который может затем использоваться в *box.rollback_to_savepoint(savepoint)*. Точки сохранения могут быть созданы, пока активна транзакция, и удаляются после окончания транзакции.

**возвращает** таблица точки сохранения

**тип возвращаемого значения** Lua-объект

**возвращает** ошибка, если точку сохранения нельзя указать в отсутствие активной транзакции.

**Возможные ошибки:** ошибка, если по какой-либо причине нельзя выделить память.

**Example**

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }

-- Rollback the transaction to a savepoint --
box.begin()
box.space.bands:insert { 4, 'The Beatles', 1960 }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
save1 = box.savepoint()
box.space.bands:replace { 1, 'Pink Floyd', 1965 }
box.rollback_to_savepoint(save1)
box.commit()
```

### box.rollback_to_savepoint()

box.rollback_to_savepoint(*savepoint*)

Запрещение окончания транзакции, но отмена всех изменений и операций *box.savepoint()*, сделанных после точки сохранения.

**возвращает** ошибка, если точка сохранения не может быть установлена при отсутствии активной транзакции.

**Возможные ошибки:** ошибка, если отсутствует точка сохранения.

**Example**

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }

-- Rollback the transaction to a savepoint --
box.begin()
box.space.bands:insert { 4, 'The Beatles', 1960 }
save1 = box.savepoint()
box.space.bands:replace { 1, 'Pink Floyd', 1965 }
box.rollback_to_savepoint(save1)
box.commit()
```

### box.atomic()

box.atomic([*opts*], *tx-function*[, *function-arguments*])

Выполнение функции так, как будто функция начинается с явного вызова *box.begin()* и заканчивается неявным вызовом *box.commit()* после успешного выполнения или же заканчивается неявным вызовом *box.rollback()* в случае ошибки.

**Параметры**

- `opts` (*table*) – (optional) transaction options:
  - `txn_isolation` – the *transaction isolation level*
  - `timeout` – a timeout (in seconds), after which the transaction is rolled back
- `tx-function` (*string*) – the function name
- `function-arguments` – (optional) arguments passed to the function

**возвращает** the result of the function passed to `atomic()` as an argument

**Возможные ошибки:**

- ошибка и прерывание транзакции в случае конфликта.

- error and abort the transaction if the timeout is exceeded.
- ошибка, если операция не может выполнить запись на диск.
- ошибка, если по какой-либо причине нельзя выделить память.

**Example**

```
-- Create an index with the specified sequence --
box.schema.sequence.create('id_sequence', { min = 1 })
box.space.bands:create_index('primary', { parts = { 'id' }, sequence = 'id_sequence
↪' })

-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }

-- Define a function --
local function insert_band(band_name, year)
    box.space.bands:insert { nil, band_name, year }
end

-- Begin and commit the transaction implicitly --
box.atomic(insert_band, 'The Beatles', 1960)

-- Begin the transaction with the specified isolation level --
box.atomic({ txn_isolation = 'read-committed' },
        insert_band, 'The Rolling Stones', 1962)
```

### box.on_commit()

box.on_commit(*trigger-function*[, *old-trigger-function*])

Определения триггера, выполняемого в случае окончания транзакции в связи с *box.commit()*.

Функция с триггером может принимать параметр с итератором, как описано в примере к данному разделу.

Функция с триггером не должна получать доступ к любым спейсам базы данных.

Если триггер не сработает и выдаст ошибку, результат будет неблагоприятным, чего следует избегать – используйте Lua-механизм `pcall()` вокруг кода, который может не сработать.

`box.on_commit()` следует вызывать в пределах транзакции, и триггер прекращает существование по окончании транзакции.

**Параметры**

- `trigger-function` (`function`) – функция, в которой будет триггер
- `old-trigger-function` (`function`) – существующая функция с триггером, которую заменит новая

**возвращает** nil или указатель функции

Если указаны параметры `(nil, old-trigger-function)`, старый триггер будет удален.

Подробная информация о характеристиках триггера находится в разделе *Триггеры*.

**Example 1**

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }

-- Define a function called on commit --
function print_commit_result()
    print('Commit happened')
end

-- Commit the transaction --
box.begin()
box.space.bands:insert { 4, 'The Beatles', 1960 }
box.on_commit(print_commit_result)
box.commit()
```

**Example 2**

The function parameter can be an iterator. The iterator goes through the effects of every request that changed a space during the transaction.

The iterator has:

- an ordinal request number
- the old value of the tuple before the request (`nil` for an `insert` request)
- the new value of the tuple after the request (`nil` for a `delete` request)
- the ID of the space

The example below displays the effects of two `replace` requests:

```
-- Insert test data --
box.space.bands:insert { 1, 'Roxette', 1986 }
box.space.bands:insert { 2, 'Scorpions', 1965 }
box.space.bands:insert { 3, 'Ace of Base', 1987 }

-- Define a function called on commit --
function print_replace_details(iterator)
    for request_number, old_tuple, new_tuple, space_id in iterator() do
        print('request_number: ' .. tostring(request_number))
        print('old_tuple: ' .. tostring(old_tuple))
        print('new_tuple: ' .. tostring(new_tuple))
        print('space_id: ' .. tostring(space_id))
    end
end

-- Commit the transaction --
box.begin()
box.space.bands:replace { 1, 'The Beatles', 1960 }
box.space.bands:replace { 2, 'The Rolling Stones', 1965 }
box.on_commit(print_replace_details)
box.commit()
```

The output might look like this:

```
request_number: 1
old_tuple: [1, 'Roxette', 1986]
new_tuple: [1, 'The Beatles', 1960]
space_id: 512
request_number: 2
old_tuple: [2, 'Scorpions', 1965]
new_tuple: [2, 'The Rolling Stones', 1965]
space_id: 512
```

### box.on_rollback()

box.on_rollback(*trigger-function*[, *old-trigger-function*])
Определение триггера, выполняемого по окончании транзакции в связи с *box.rollback()*.

Используются точно такие же параметры и предупреждения, как в *box.on_commit()*.

### box.is_in_txn()

box.is_in_txn()
В процессе транзакции (например, пользователь вызвал *box.begin()* и еще не вызвал ни *box.commit*, ни *box.rollback()*) возвращается `true`. В остальных случаях возвращается `false`.

### Functions for SQL

The `box` module contains some functions related to SQL:

- `box.schema.func.create` – for making Lua functions callable from SQL statements. See *Calling Lua routines from SQL* in the *SQL Plus Lua* section.
- `box.execute` – for making SQL statements callable from Lua functions. See the *SQL user guide*.
- `box.prepare` and `box.unprepare`.

Some SQL statements are illustrated in the *SQL tutorial*.

Below is a list of all SQL functions and members.

| Имя | Назначение |
|---|---|
| *box.execute()* | Make Lua functions callable from SQL statements. See *Calling Lua routines from SQL* in the *SQL Plus Lua* section |
| *box.prepare()* | Make SQL statements callable from Lua functions. See the *SQL user guide* |
| *object prepared_table* | Methods for prepared SQL statement |

### box.execute()

box.execute(*sql-statement*[, *extra-parameters*])

Execute the SQL statement contained in the `sql-statement` parameter.

**Параметры**

- `sql-statement` (*string*) – statement, which should conform to *the rules for SQL grammar*
- `extra-parameters` (*table*) – optional table for placeholders in the statement

**возвращает** depends on statement

There are two ways to pass extra parameters to `box.execute()`:

- The first way, which is the preferred way, is to put placeholders in the string, and pass a second argument, an `extra-parameters` table. A placeholder is either a question mark «?», or a colon «:» followed by a name. An extra parameter is any Lua expression.

  If placeholders are question marks, then they are replaced by `extra-parameters` values in corresponding positions. That is, the first `?` is replaced by the first extra parameter, the second `?` is replaced by the second extra parameter, and so on.

  If placeholders are `:names`, then they are replaced by `extra-parameters` values with corresponding names.

  For example, this request that contains literal values `1` and `'x'`:

  ```
  box.execute([[INSERT INTO tt VALUES (1, 'x');]]);
  ```

  ... is the same as the request below containing two question-mark placeholders (`?` and `?`) and a two-element `extra-parameters` table:

  ```
  x = {1,'x'}
  box.execute([[INSERT INTO tt VALUES (?, ?);]], x);
  ```

  ... and is the same as this request containing two `:name` placeholders (`:a` and `:b`) and a two-element `extra-parameters` table with elements named «a» and «b»:

  ```
  box.execute([[INSERT INTO tt VALUES (:a, :b);]], {{[':a']=1},{[':b']='x'}})
  ```

- The second way is to concatenate strings. For example, the Lua script below inserts 10 rows with different primary-key values into table `t`:

  ```
  for i=1,10,1 do
      box.execute("insert into t values (" .. i .. ")")
  end
  ```

  When creating SQL statements based on user input, application developers should beware of SQL injection.

Since `box.execute()` is an invocation of a Lua function, it either causes an error message or returns a value.

For some statements the returned value contains a field named `rowcount`, for example:

```
tarantool> box.execute([[CREATE TABLE table1 (column1 INT PRIMARY key, column2␣
↪VARCHAR(10));]])
---
- rowcount: 1
...
tarantool> box.execute([[INSERT INTO table1 VALUES (55,'Hello SQL world!');]])
---
- rowcount: 1
...
```

For statements that cause generation of values for PRIMARY KEY AUTOINCREMENT columns, there is a field named `autoincrement_id`.

For SELECT or PRAGMA statements, the returned value is a *result set*, containing a field named `metadata` (a table with column names and Tarantool/NoSQL type names) and a field named `rows` (a table with the contents of each row).

For example, for a statement `SELECT "x" FROM t WHERE "x"=5;` where `"x"` is an INTEGER column and there is one row, a display on the Tarantool client might look like this:

```
tarantool> box.execute([[SELECT "x" FROM t WHERE "x"=5;]])
---
- metadata:
  - name: x
    type: integer
  rows:
  - [5]
...
```

For a look at raw format of SELECT results, see *Binary protocol – responses for SQL*.

The order of components within a map is not guaranteed.

If `sql_full_metadata` in the *_session_settings* system table is TRUE, then result set metadata may include these things in addition to `name` and `type`:

- `collation` (present only if COLLATE clause is specified for a STRING) = *«Collation»*.
- `is_nullable` (present only if the *select list* specified a base table column and nothing else) = false if column was defined as *NOT NULL*, otherwise true. If this is not present, that implies that nullability is unknown.
- `is_autoincrement` (present only if the select list specified a base table column and nothing else) = true if column was defined as *PRIMARY KEY AUTOINCREMENT*, otherwise false.
- `span` (always present) = the original expression in a select list, which often is the same as `name` if the select list specifies a column name and nothing else, but otherwise differs, for example, after `SELECT x+55 AS x FROM t;` the `name` is X and the `span` is x+55. If `span` and `name` are the same then the content is MP_NIL.

Alternative: if you are using the Tarantool server as a client, you can switch languages as follows:

```
\set language sql
\set delimiter ;
```

Afterwards, you can enter any SQL statement directly without needing `box.execute()`.

There is also an `execute()` function available in *module net.box*. For example, you can execute `conn:execute(sql-statement])` after `conn = net_box.connect(url-string)`.

### box.prepare()

box.prepare(*sql-statement*)

Prepare the SQL statement contained in the sql-statement parameter. The syntax and requirements for box.prepare are the same as for *box.execute()*.

**Параметры**

- sql-statement (*string*) – statement, which should conform to *the rules for SQL grammar*

**возвращает** prepared_table, with id and methods and metadata

**тип возвращаемого значения** таблица

box.prepare compiles an SQL statement into byte code and saves the byte code in a cache. Since compiling takes a significant amount of time, preparing a statement will enhance performance if the statement is executed many times.

If box.prepare succeeds, prepared_table contains:

- stmt_id: integer – an identifier generated by a hash of the statement string
- execute: function
- params: map [name : string, type : string] – parameter descriptions
- unprepare: function
- metadata: map [name : string, type : string] (This is present only for SELECT or PRAGMA statements and has the same contents as the *result set metadata* for box.execute)
- param_count: integer – number of parameters

This can be used by *prepared_table:execute()* and by *prepared_table:unprepare()*.

The prepared statement cache (which is also called the prepared statement holder) is «shared», that is, there is one cache for all sessions. However, session X cannot execute a statement prepared by session Y. For monitoring the cache, see *box.info().sql*. For changing the cache, see *(Configuration reference) sql_cache_size*.

Prepared statements will «expire» (become invalid) if any database object is dropped or created or altered – even if the object is not mentioned in the SQL statement, even if the create or drop or alter is rolled back, even if the create or drop or alter is done in a different session.

### object prepared_table

object prepared_table

prepared_table:execute([*extra-parameters*])

Execute a statement that has been prepared with *box.prepare()*.

Parameter prepared_table should be the result from box.prepare().

Parameter extra-parameters should be an optional table to match *placeholders or named parameters* in the statement.

There are two ways to execute: with the method or with the statement id. That is, *prepared_table*:execute() and box.execute(*prepared_table*.stmt_id) do the same thing.

Example: here is a test. This function inserts a million rows in a table using a prepared INSERT statement.

```
function f()
  local p, start_time
  box.execute([[DROP TABLE IF EXISTS t;]])
  box.execute([[CREATE TABLE t (s1 INTEGER PRIMARY KEY);]])
  start_time = os.time()
  p = box.prepare([[INSERT INTO t VALUES (?);]])
  for i=1,1000000 do p:execute({i}) end
  p:unprepare()
  end_time = os.time()
  box.execute([[COMMIT;]])
  print(end_time - start_time) -- elapsed time
end
f()
```

Take note of the elapsed time. Now change the line with the loop to: `for i=1,1000000 do box.execute([[INSERT INTO t VALUES (?);]], {i}) end` Run the function again, and take note of the elapsed time again. The function which executes the prepared statement will be about 15% faster, though of course this will vary depending on Tarantool version and environment.

`prepared_table:unprepare()`

Undo the result of an earlier *box.prepare()* request. This is equivalent to standard-SQL DEALLOCATE PREPARE.

Parameter `prepared_table` should be the result from `box.prepare()`.

There are two ways to unprepare: with the method or with the statement id. That is, *prepared_table*`:unprepare()` and `box.unprepare(`*prepared_table*`.stmt_id)` do the same thing.

Tarantool strongly recommends using `unprepare` as soon as the immediate objective (executing a prepared statement multiple times) is done, or whenever a prepared statement expires. There is no automatic eviction policy, although automatic `unprepare` will happen when the session disconnects (the session's prepared statements will be removed from the prepared-statement cache).

### Event watchers

Since *2.10.0*.

The `box` module contains some features related to event subscriptions, also known as *watchers*. The subscriptions are used to inform the client about server-side *events*. Each event subscription is defined by a certain key.

**Event** An event is a state change or a system update that triggers the action of other systems. To read more about built-in events in Tarantool, check the *system events* section.

**State** A state is an internally stored key-value pair. The key is a string. The value is an arbitrary type that can be encoded as MsgPack. To update a state, use the `box.broadcast()` function.

**Watcher** A watcher is a *callback* that is invoked when a state change occurs. To register a local watcher, use the `box.watch()` function. To create a remote watcher, use the `watch()` function from the `net.box` module. Note that it is possible to register more than one watcher for the same key.

### How a watcher works

First, you register a watcher. After that, the watcher callback is invoked for the first time. In this case, the callback is triggered whether or not the key has already been broadcast. All subsequent invocations are triggered with *box.broadcast()* called on the remote host. If a watcher is subscribed for a key that has not been broadcast yet, the callback is triggered only once, after the registration of the watcher.

The watcher callback takes two arguments. The first argument is the name of the key for which it was registered. The second one contains current key data. The callback is always invoked in a new fiber. It means that it is allowed to yield in it. A watcher callback is never executed in parallel with itself. If the key is updated while the watcher callback is running, the callback will be invoked again with the new value as soon as it returns.

`box.watch` and `box.broadcast` functions can be used before *box.cfg*.

Below is a list of all functions and pages related to watchers or events.

| Name | Use |
|---|---|
| *box.watch()* | Create a local watcher. |
| *conn:watch()* | Create a watcher for the remote host. |
| *box.broadcast()* | Update a state. |
| *Built-in events* | Predefined events in Tarantool |

### box.watch()

`box.watch`(*key*, *func*)

Subscribe to events broadcast by a local host.

**Параметры**

- `key` (*string*) – key name of the event to subscribe to
- `func` (`function`) – callback to invoke when the key value is updated

**Return** a watcher handle. The handle consists of one method – `unregister()`, which unregisters the watcher.

To read more about watchers, see the *Functions for watchers* section.

---

**Примечание:** Keep in mind that garbage collection of a watcher handle doesn't lead to the watcher's destruction. In this case, the watcher remains registered. It is okay to discard the result of `watch` function if the watcher will never be unregistered.

---

**Example:**

```
-- Broadcast value 42 for the 'foo' key.
box.broadcast('foo', 42)

local log = require('log')
-- Subscribe to updates of the 'foo' key.
local w = box.watch('foo', function(key, value)
    assert(key == 'foo')
    log.info("The box.id value is '%d'", value)
end)
```

If you don't need the watcher anymore, you can unregister it using the command below:

```
w:unregister()
```

### box.broadcast()

box.broadcast(*key*, *value*)

Update the value of a particular key and notify all key watchers of the update.

**Параметры**

- `key` (*string*) – key name of the event to subscribe to
- `value` – any data that can be encoded in MsgPack

**Return** none

**Possible errors:**

- The value can't be encoded as MsgPack.
- The key refers to a `box.` system event

**Example:**

```
-- Broadcast value 42 for the 'foo' key.
box.broadcast('foo', 42)
```

### System events

Since *2.10.0*.

Predefined events have a special naming schema – theirs names always start with the reserved `box.` prefix. It means that you cannot create new events with it.

The system processes the following events:

- `box.id`
- `box.status`
- `box.election`
- `box.schema`
- `box.shutdown`

In response to each event, the server sends back certain `IPROTO` fields.

The events are available from the beginning as non-*MP_NIL*. If a watcher subscribes to a system event before it has been broadcast, it receives an empty table for the event value.

The event is generated when there is a change in any of the values listed in the event. For example, see the parameters in the `box.id` event below – `id`, `instance_uuid`, and `replicaset_uuid`. Suppose the `id` value (`box.info.id`) has changed. This triggers the `box.info` event, which states that the value of `box.info.id` has changed, while `box.info.uuid` and `box.info.cluster.uuid` remain the same.

### box.id

Contains *identification* of the instance. Value changes are rare.

- `id`: the numeric instance ID is unknown before the registration. For anonymous replicas, the value is `0` until they are officially registered.
- `instance_uuid`: the UUID of the instance never changes after the first *box.cfg*. The value is unknown before the `box.cfg` call.
- `replicaset_uuid`: the value is unknown until the instance joins a replicaset or boots a new one.

```
-- box.id value
{
MP_STR “id”: MP_UINT; box.info.id,
MP_STR “instance_uuid”: MP_UUID; box.info.uuid,
MP_STR “replicaset_uuid”: MP_UUID box.info.cluster.uuid,
}
```

### box.status

Contains generic information about the instance status.

- `is_ro`: *indicates the read-only mode* or the `orphan` status.
- `is_ro_cfg`: indicates the *read_only* mode for the instance.
- `status`: shows the status of an instance.

```
{
MP_STR “is_ro”: MP_BOOL box.info.ro,
MP_STR “is_ro_cfg”: MP_BOOL box.cfg.read_only,
MP_STR “status”: MP_STR box.info.status,
}
```

### box.election

Contains fields of *box.info.election* that are necessary to find out the most recent writable leader.

- `term`: shows the current election term.
- `role`: indicates the election state of the node – `leader`, `follower`, or `candidate`.
- `is_ro`: *indicates the read-only mode* or the `orphan` status.
- `leader`: shows the leader node ID in the current term.

```
{
MP_STR “term”: MP_UINT box.info.election.term,
MP_STR “role”: MP_STR box.info.election.state,
MP_STR “is_ro”: MP_BOOL box.info.ro,
MP_STR “leader”: MP_UINT box.info.election.leader,
}
```

### box.schema

Contains schema-related data.

- `version`: shows the schema version.

```
{
MP_STR "version": MP_UINT schema_version,
}
```

### box.shutdown

Contains a boolean value which indicates whether there is an active shutdown request.

The event is generated when the server receives a shutdown request (`os.exit()` command or *SIGTERM* signal).

The `box.shutdown` event is applied for the graceful shutdown protocol. It is a feature which is available since *2.10.0*. This protocol is supposed to be used with connectors to signal a client about the upcoming server shutdown and close active connections without broken requests. For more information, refer to the *graceful shutdown protocol* section.

### Usage example

```
local conn = net.box.connect(URI)
local log = require('log')
-- Subscribe to updates of key 'box.id'
local w = conn:watch('box.id', function(key, value)
    assert(key == 'box.id')
    log.info("The box.id value is '%s'", value)
end)
```

If you want to unregister the watcher when it's no longer needed, use the following command:

```
w:unregister()
```

### Функция box.once

box.once(*key*, *function*[, ...])
: Выполнение функции при условии, что она раньше не выполнялась. Передаваемое значение проверяется на предмет того, выполнялась ли функция. Если она выполнялась, ничего не происходит. В противном случае вызывается функция.

  См. пример использования `box.once()` во время *настройки набора реплик*.

  Если в `box.once()` возникает ошибка во время инициализации базы данных, можно повторно запустить невыполненный блок `box.once()`, не останавливая базу данных. Для этого удалите объект `once` из системного спейса *_schema*. Введите команду `box.space._schema:select{}`, найдите объект `once` и удалите его. Например, повторное выполнение блока `key='hello'` :

  Когда `box.once()` используется для инициализации, следует подождать, пока база данных не будет в нужном состоянии (только для чтения или для чтения и записи). Для этого см. функции во *Вложенный модуль box.ctl*.

```
tarantool> box.space._schema:select{}
---
- - ['cluster', 'b4e15788-d962-4442-892e-d6c1dd5d13f2']
  - ['max_id', 512]
  - ['oncebye']
  - ['oncehello']
  - ['version', 1, 7, 2]
...

tarantool> box.space._schema:delete('oncehello')
---
- ['oncehello']
...

tarantool> box.once('hello', function() end)
---
...
```

**Параметры**

- `key` (*string*) – значение для проверки
- `function` (`function`) – функция
- `...` – аргументы, которые следует передать в функцию

---

**Примечание:** Параметр `key` сохраняется в системном спейсе *_schema* после вызова `box.once()`, чтобы предотвратить повторный вызов по ключу. Эти ключи распространяются на набор реплик. Поэтому одновременный вызов `box.once` с одинаковыми ключами на двух экземплярах одного набора реплик может быть успешным, но приведет к конфликту транзакций.

---

### Функция box.snapshot

`box.snapshot()`

**Memtx**

Take a snapshot of all data and store it in *snapshot.dir*`/<latest-lsn>.snap`. To take a snapshot, Tarantool first enters the delayed garbage collection mode for all data. In this mode, the *Tarantool garbage collector* will not remove files which were created before the snapshot started, it will not remove them until the snapshot has finished. To preserve consistency of the primary key, used to iterate over tuples, a copy-on-write technique is employed. If the master process changes part of a primary key, the corresponding process page is split, and the snapshot process obtains an old copy of the page. In effect, the snapshot process uses multi-version concurrency control in order to avoid copying changes which are superseded while it is running.

Since a snapshot is written sequentially, you can expect a very high write performance (averaging to 80MB/second on modern disks), which means an average database instance gets saved in a matter of minutes. You may restrict the speed by changing *snapshot.snap_io_rate_limit*.

---

**Примечание:** При условии, что происходят изменения в родительском индексе в ходе многопоточного обновления данных, будет происходить и расщепление страниц, поэтому возникнет необходимость в наличии дополнительной свободной памяти для выполнения этой команды. В

среднем, будет достаточно 10% от *memtx_memory*. Оператор подождет окончания создания снимка и вернет результат операции.

---

**Примечание: Обновление:** До версии 1.6.6 Tarantool процесс создания снимка вызывал клонирование системного процесса (fork), что могло привести к скачкам задержки отклика. Начиная с версии 1.6.6 Tarantool, процесс создания снимка создает вид постоянного просмотра, который и записывается в файл снимка с помощью отдельного потока (поток упреждающей записи в журнал).

Although `box.snapshot()` does not cause a fork, there is a separate fiber which may produce snapshots at regular intervals – see the discussion of the *checkpoint daemon*.

---

**Пример:**

```
tarantool> box.info.version
---
- 1.7.0-1216-g73f7154
...
tarantool> box.snapshot()
---
- ok
...
tarantool> box.snapshot()
---
- error: can't save snapshot, errno 17 (File exists)
...
```

Создание снимка не приводит к записи нового журнала упреждающей записи на сервере. После создания снимка старые WAL-файлы можно удалить, если все реплицируемые данные актуальны. Но WAL-файл на момент начала работы `box.snapshot()` следует сохранить на случай восстановления, поскольку он содержит записи журнала после начала работы `box.snapshot()`.

Другим способом сохранения снимка будет отправка сигнала SIGUSR1 процессу. Хотя это может быть удобно, не рекомендуется использовать такой метод в автоматическом процессе: сигнал не дает возможность проверить, был ли корректно сделан снимок.

### Vinyl

При использовании vinyl'а вставляемые данные складываются в память до тех пор, пока не будет достигнут предел, установленный в параметре *vinyl_memory*. Затем vinyl автоматически делает дамп на диск. `box.snapshot()` форсирует создание дампа, чтобы иметь возможность восстановить данные из этой контрольной точки. Файлы снимков хранятся в *space_id*/*index_id*/`*.run`. Таким образом, строго все данные, которые были записаны во время LSN контрольной точки, находятся в `*.run` файлах на диске, а все операции, которые происходили после контрольной точки, будут записаны в `*.xlog`. Все файлы дампа, созданные функцией `box.snapshot()`, консистентны и имеют тот же LSN, что и контрольная точка.

На контрольной точке vinyl также пересматривает журнал метаданных `*.vylog`, содержащий операции манипуляции с данными, такие как «создать файл» и «удалить файл». Он проходит по логу, удаляет дублирующие операции из памяти и создает новый файл `*.vylog`, присваивая ему имя в соответствии с *vclock* новой контрольной точки, оставляя только операци создания. Эта процедура очищает `*.vylog` и полезна для восстановления, так как имя лога совпадает с именем подписи контрольной точки.

### Константа box.NULL

Имеется целый ряд серьезных проблем при использовании значения **nil** из Lua в таблицах. Например: вы не можете корректно оценить длину таблицы, не являющейся последовательностью. (Узнайте больше о типах данных в Lua и LuaJIT.)

**Пример:**

```
tarantool> t = {0, nil, 1, 2, nil}
---
...

tarantool> t
---
- - 0
  - null
  - 1
  - 2
...

tarantool> #t
---
- 4
...
```

Вывод в консоль `t` обрабатывает значения **nil** в середине и в конце таблицы по-разному. Это вызвано неопределённым поведением.

---

**Примечание:** Попытка найти длину для разреженного массива в LuaJIT приводит к другому случаю неопределённого поведения.

---

Для избежания этой проблемы используйте имеющуюся в Tarantool константу `box.NULL` вместо значения **nil**. `box.NULL` является местозаполнителем для значения **nil** в таблицах с целью сохранения ключа без значения.

### Использование box.NULL

`box.NULL` является значением типа cdata, представляющим нулевой указатель (NULL pointer). Оно подобно `msgpack.NULL`, `json.NULL` и `yaml.NULL`. Таким образом, оно является некоторым не **nil** значением, даже если является указателем на NULL.

Используйте `box.NULL` только с NULL, написанным заглавными буквами (`box.null` является ошибкой).

---

**Примечание:** Технически, `box.NULL` соответствует `ffi.cast('void *', 0)`.

---

**Пример:**

```
tarantool> t = {0, box.NULL, 1, 2, box.NULL}
---
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> t
---
- - 0
  - null # cdata
  - 1
  - 2
  - null # cdata
...

tarantool> #t
---
- 5
...
```

**Примечание:** Заметьте, что `t[2]` демонстрирует один и тот же вывод `null` в обоих примерах. Однако, в данном примере `t[2]` и `t[5]` являются типом cdata, в то время как в предыдущем примере их тип был **nil**.

**Важно:** Избегайте использования неявных сравнений с обнуляемыми (nullable) значениями при использовании `box.NULL`. В связи со штатным поведением Lua, возвращение любого результата, кроме **false** (ложь) или **nil** (ничто), из выражения условия считается возвращением **true** (истина). Как и упоминалось ранее, `box.NULL` является указателем.

Поэтому выражение `box.NULL` всегда будет расцениваться как **true** (истина) в случае использования в качестве условия в сравнении. Это означает, что код

```
if box.NULL then func() end
```

всегда будет выполнять функцию `func()` (потому, что условие `box.NULL` всегда будет не **false** (ложь) и не **nil** (ничто)).

### Distinction of nil and box.NULL

Используйте выражение `x == nil` для проверки того, является ли `x` **nil** или `box.NULL`.

Для выяснения того, является ли `x` в действительности **nil**, но не `box.NULL`, используйте следующее условие:

```
type(x) == 'nil'
```

Если оно истинно (**true**), то `x` – это **nil**, но не``box.NULL``.

Вы можете использовать следующее выражение для `box.NULL`:

```
x == nil and type(x) == 'cdata'
```

Если вышеуказанное выражение истинно (**true**), то `x` – это `box.NULL`.

**Примечание:** Конвертируя данные в различные форматы (JSON, YAML, msgpack), вы должны ожидать возможного преобразования всех **nil** в разреженных массивах в `box.NULL`. Стоит ответить,

что конвертация может происходить неожиданно (например: при отправке данных через *net.box* или при получении данных из *спейсов* и т.п.).

```
tarantool> type(({1, nil, 2})[2])
---
- nil
...

tarantool> type(json.decode(json.encode({1, nil, 2}))[2])
---
- cdata
...
```

Вы должны ожидать подобное поведение и использовать соответствующее выражение условия. Используйте явное сравнение `x == nil` для проверки на отсутствующее значение (NULL) в обнуляемых (nullable) переменных. Оно позволит обнаружить как **nil**, так и `box.NULL`.

### 9.5.2 Модуль buffer

Модуль `buffer` возвращает буфер, допускающий динамическое изменение размера, который используется только в качестве опции для методов *модуля net.box* или *модуля msgpack*.

Как правило, модуль `net.box` возвращает Lua-таблицу. Если используется опция `buffer`, то методы модуля `net.box` возвращают неформатированную строку *строку MsgPack*. Это экономит время работы на сервере, если в клиентском приложении есть собственная процедура декодирования MsgPack-строк.

Буфер использует четыре указателя для управления его мощностью:

- `buf` – указатель на начало буфера
- `rpos` – указатель на начало участка памяти, доступного для чтения данных («read position»)
- `wpos` – указатель на конец участка памяти для чтения и начало участка для записи данных («write position»)
- `epos` – указатель на конец участка для записи данных («end position»)

`buffer.ibuf()`
Создать новый буфер.

**Пример:**

В этом примере мы покажем, что использование буфера позволит вам сохранить данные в том же формате, в котором они пришли с сервера. Так что если получить данные с сервера нужно только для отправки куда-то дальше, то с буфером это будет гораздо быстрее.

```
box.cfg{listen = 3301}
buffer = require('buffer')
net_box = require('net.box')
msgpack = require('msgpack')

box.schema.space.create('tester')
box.space.tester:create_index('primary')
box.space.tester:insert({1, 'ABCDE', 12345})

box.schema.user.create('usr1', {password = 'pwd1'})
```

(continues on next page)

(продолжение с предыдущей страницы)

```
box.schema.user.grant('usr1', 'read,write,execute', 'space', 'tester')

ibuf = buffer.ibuf()

conn = net_box.connect('usr1:pwd1@localhost:3301')
conn.space.tester:select({}, {buffer=ibuf})

msgpack.decode_unchecked(ibuf.rpos)
```

Результат последнего запроса выглядит следующим образом:

```
tarantool> msgpack.decode_unchecked(ibuf.rpos)
---
- {48: [['ABCDE', 12345]]}
- 'cdata<char *>: 0x7f97ba10c041'
...
```

---

**Примечание:** До версии 1.7.7 Tarantool в данном случае следует использовать функцию `msgpack.ibuf_decode(ibuf.rpos)`. Начиная с версии 1.7.7 Tarantool , `ibuf_decode` объявлена устаревшей.

---

`object buffer_object`

`buffer_object:alloc(`*size*`)`
Аллоцировать `size` байтов для `buffer_object`'а.

**Параметры**

- `size` (`number`) – количество байтов для аллоцирования

**возвращает** `wpos`

`buffer_object:capacity()`
Вернуть мощность `buffer_object`'а.

**возвращает** `epos - buf`

`buffer_object:checksize(`*size*`)`
Проверить, доступно ли `size` байтов для чтения из `buffer_object`'а.

**Параметры**

- `size` (`number`) – память в байтах для проверки

**возвращает** `rpos`

`buffer_object:pos()`
Вернуть размер участка, занятого данными.

**возвращает** `rpos - buf`

`buffer_object:read(`*size*`)`
Прочитать `size` байтов из буфера.

`buffer_object:recycle()`
Очистить слоты памяти, выделенные для `buffer_object`'а.

```
tarantool> ibuf:recycle()
---
...
tarantool> ibuf.buf, ibuf.rpos, ibuf.wpos, ibuf.epos
---
- 'cdata<char *>: NULL'
- 'cdata<char *>: NULL'
- 'cdata<char *>: NULL'
- 'cdata<char *>: NULL'
...
```

`buffer_object:reset()`
: Очистить слоты памяти, использованные `buffer_object`'ом. Этот метод позволяет сохранить буфер, но убрать из него все данные. Это полезно, если вы собираетесь использовать буфер дальше.

```
tarantool> ibuf:reset()
---
...
tarantool> ibuf.buf, ibuf.rpos, ibuf.wpos, ibuf.epos
---
- 'cdata<char *>: 0x010cc28030'
- 'cdata<char *>: 0x010cc28030'
- 'cdata<char *>: 0x010cc28030'
- 'cdata<char *>: 0x010cc2c000'
...
```

`buffer_object:reserve(`*size*`)`
: Зарезервировать память для `buffer_object`. Проверить, достаточно ли памяти, чтобы записать `size` байтов после `wpos`. Если нет, `epos` будет сдвигаться, пока `size` байтов не будет доступно.

`buffer_object:size()`
: Вернуть участок, доступный для чтения данных.

  **возвращает** `wpos - rpos`

`buffer_object:unused()`
: Вернуть участок, доступный для записи данных.

  **возвращает** `epos - wpos`

### Модуль buffer и skip_header

В примере из предыдущего раздела

```
tarantool> msgpack.decode_unchecked(ibuf.rpos)
---
- {48: [['ABCDE', 12345]]}
- 'cdata<char *>: 0x7f97ba10c041'
...
```

было показано, что обычно net.box ответ включает в себя заголовок – 48 (в шестнадцатиричной системе – 30), который является *ключом* для IPROTO_DATA. Но в некоторых ситуациях, например, при передаче буфера в функцию C, которая ожидает массив байт MsgPack без заголовка, заголовок можно пропустить. Это делается путем указания `skip_header=true` в качестве опции *conn.space.space-name:select{...}* или *conn.space.space-name:insert{...}* или *conn.space.space-*

*name:replace{...}* или *conn.space.space-name:update{...}* или *conn.space.space-name:upsert{...}* или *conn.space.space-name:delete{...}*. По умолчанию `skip_header=false`.

Ниже приведен конец того же примера, но с использованием `skip_header=true`.

```
ibuf = buffer.ibuf()

conn = net_box.connect('usr1:pwd1@localhost:3301')
conn.space.tester:select({}, {buffer=ibuf, skip_header=true})

msgpack.decode_unchecked(ibuf.rpos)
```

Результат последнего запроса выглядит следующим образом:

```
tarantool> msgpack.decode_unchecked(ibuf.rpos)
---
- {48: [['ABCDE', 12345]]}
- 'cdata<char *>: 0x7f97ba10c041'
...
```

Заметьте, что заголовок IPROTO_DATA (48) ушел.

Результат остается внутри массива, что видно из того, что он заключен в квадратные скобки. Пропустить заголовок массива можно и функцией `msgpack.decode_array_header()`.

### 9.5.3 Module checks

**Since:** *2.11.0*

The `checks` module provides the ability to check the types of arguments passed to a Lua function. You need to call the *checks(type_1, ...)* function inside the target Lua function and pass *one or more* type qualifiers to check the corresponding argument types. There are two types of type qualifiers:

- A *string type qualifier* checks whether a function's argument conforms to the specified type. Example: `'string'`.
- A *table type qualifier* checks whether the values of a table passed as an argument conform to the specified types. Example: `{ 'string', 'number' }`.

---

**Примечание:** For earlier versions, you can install the `checks` module from the Tarantool rocks repository.

---

#### Loading checks

In Tarantool *2.11.0* and later versions, the *checks API* is available in a script without loading the module.

For earlier versions, you need to install the `checks` module from the Tarantool rocks repository and load the module using the `require()` directive:

```
local checks = require('checks')
```

### Number of arguments to check

For each argument to check, you need to specify its own type qualifier in the *checks(type_1, ...)* function.

### One argument

In the example below, the `checks` function accepts a `string` type qualifier to verify that only a string value can be passed to the `greet` function. Otherwise, an error is raised.

```
function greet(name)
    checks('string')
    return 'Hello, ' .. name
end
--[[
greet('John')
-- returns 'Hello, John'

greet(123)
-- raises an error: bad argument #1 to nil (string expected, got number)
--]]
```

### Multiple arguments

To check the types of several arguments, you need to pass the corresponding type qualifiers to the `checks` function. In the example below, both arguments should be string values.

```
function greet_fullname(firstname, lastname)
    checks('string', 'string')
    return 'Hello, ' .. firstname .. ' ' .. lastname
end
--[[
greet_fullname('John', 'Smith')
-- returns 'Hello, John Smith'

greet_fullname('John', 1)
-- raises an error: bad argument #2 to nil (string expected, got number)
--]]
```

To skip checking specific arguments, use the *? placeholder*.

### Variable number of arguments

You can check the types of explicitly specified arguments for functions that accept a variable number of arguments.

```
function extra_arguments_num(a, b, ...)
    checks('string', 'number')
    return select('#', ...)
end
--[[
```

(continues on next page)

(продолжение с предыдущей страницы)

```
extra_arguments_num('a', 2, 'c')
-- returns 1

extra_arguments_num('a', 'b', 'c')
-- raises an error: bad argument #1 to nil (string expected, got number)
--]]
```

### String type qualifier

This section describes how to check a specific argument type using a string type qualifier:

- The *Supported types* section describes all the types supported by the `checks` module.
- If required, you can make a *union type* to allow an argument to accept several types.
- You can make any of the supported types *optional*.
- To skip checking specific arguments, use the *? placeholder*.

### Supported types

#### Lua types

A string type qualifier can accept any of the Lua types, for example, `string`, `number`, `table`, or `nil`. In the example below, the `checks` function accepts `string` to validate that only a string value can be passed to the `greet` function.

```
function greet(name)
    checks('string')
    return 'Hello, ' .. name
end
--[[
greet('John')
-- returns 'Hello, John'

greet(123)
-- raises an error: bad argument #1 to nil (string expected, got number)
--]]
```

#### Tarantool types

You can use *Tarantool-specific types* in a string qualifier. The example below shows how to check that a function argument is a decimal value.

```
local decimal = require('decimal')
function sqrt(value)
    checks('decimal')
    return decimal.sqrt(value)
end
--[[
```

(continues on next page)

(продолжение с предыдущей страницы)

```
sqrt(decimal.new(16))
-- returns 4

sqrt(16)
-- raises an error: bad argument #1 to nil (decimal expected, got number)
--]]
```

This table lists all the checks available for Tarantool types:

| Check | Description | See also |
|---|---|---|
| `checks('datetime')` | Check whether the specified value is *datetime_object* | *checkers.datetime(value)* |
| `checks('decimal')` | Check whether the specified value has the *decimal* type | *checkers.decimal(value)* |
| `checks('error')` | Check whether the specified value is *error_object* | *checkers.error(value)* |
| `checks('int64')` | Check whether the specified value is an `int64` value | *checkers.int64(value)* |
| `checks('interval')` | Check whether the specified value is *interval_object* | *checkers.interval(value)* |
| `checks('tuple')` | Check whether the specified value is a *tuple* | *checkers.tuple(value)* |
| `checks('uint64')` | Check whether the specified value is a `uint64` value | *checkers.uint64(value)* |
| `checks('uuid')` | Check whether the specified value is *uuid_object* | *checkers.uuid(value)* |
| `checks('uuid_bin')` | Check whether the specified value is *uuid* represented by a 16-byte binary string | *checkers.uuid_bin(value)* |
| `checks('uuid_str')` | Check whether the specified value is *uuid* represented by a 36-byte hexadecimal string | *checkers.uuid_str(value)* |

### Custom function

A string type qualifier can accept the name of a custom function that performs arbitrary validations. To achieve this, create a function returning `true` if the value is valid and add this function to the *checkers* table.

The example below shows how to use the `positive` function to check that an argument value is a positive number.

```
function checkers.positive(value)
    return (type(value) == 'number') and (value > 0)
end

function get_doubled_number(value)
    checks('positive')
    return value * 2
end
--[[
get_doubled_number(10)
-- returns 20
```

(continues on next page)

(продолжение с предыдущей страницы)

```
get_doubled_number(-5)
-- raises an error: bad argument #1 to nil (positive expected, got number)
--]]
```

### Metatable type

A string qualifier can accept a value stored in the `__type` field of the argument metatable.

```
local blue = setmetatable({ 0, 0, 255 }, { __type = 'color' })
function get_blue_value(color)
    checks('color')
    return color[3]
end
--[[
get_blue_value(blue)
-- returns 255

get_blue_value({0, 0, 255})
-- raises an error: bad argument #1 to nil (color expected, got table)
--]]
```

### Union types

To allow an argument to accept several types (a union type), concatenate type names with a pipe (`|`). In the example below, the argument can be both a `number` and `string` value.

```
function get_argument_type(value)
    checks('number|string')
    return type(value)
end
--[[
get_argument_type(1)
-- returns 'number'

get_argument_type('key1')
-- returns 'string'

get_argument_type(true)
-- raises an error: bad argument #1 to nil (number|string expected, got boolean)
--]]
```

### Optional types

To make any of the supported types optional, prefix its name with a question mark (`?`). In the example below, the `name` argument is optional. This means that the `greet` function can accept `string` and `nil` values.

```
function greet(name)
    checks('?string')
    if name ~= nil then
        return 'Hello, ' .. name
    else
        return 'Hello from Tarantool'
    end
end
--[[
greet('John')
-- returns 'Hello, John'

greet()
-- returns 'Hello from Tarantool'

greet(123)
-- raises an error: bad argument #1 to nil (string expected, got number)
--]]
```

As for a specific type, you can make a *union type* value optional: `?number|string`.

### Skipping argument checking

You can skip checking of the specified arguments using the question mark (`?`) placeholder. In this case, the argument can be any type.

```
function greet_fullname_any(firstname, lastname)
    checks('string', '?')
    return 'Hello, ' .. firstname .. ' ' .. tostring(lastname)
end
--[[
greet_fullname_any('John', 'Doe')
-- returns 'Hello, John Doe'

greet_fullname_any('John', 1)
-- returns 'Hello, John 1'
--]]
```

### Table type qualifier

A table type qualifier checks whether the values of a table passed as an argument conform to the specified types. In this case, the following checks are made:

- The argument is checked to conform to the `?table` type, and its content is validated.
- Table values are validated against the specified *string type qualifiers*.
- Table keys missing in `checks` are validated against the `nil` type.

The code below checks that the first and second table values have the `string` and `number` types.

```
function configure_connection(options)
    checks({ 'string', 'number' })
    local ip_address = options[1] or '127.0.0.1'
    local port = options[2] or 3301
    return ip_address .. ':' .. port
end
--[[
configure_connection({'0.0.0.0', 3303})
-- returns '0.0.0.0:3303'

configure_connection({'0.0.0.0', '3303'})
-- raises an error: bad argument options[2] to nil (number expected, got string)
--]]
```

In the next example, the same checks are made for the specified keys.

```
function configure_connection_opts(options)
    checks({ ip_address = 'string', port = 'number' })
    local ip_address = options.ip_address or '127.0.0.1'
    local port = options.port or 3301
    return ip_address .. ':' .. port
end
--[[
configure_connection_opts({ip_address = '0.0.0.0', port = 3303})
-- returns '0.0.0.0:3303'

configure_connection_opts({ip_address = '0.0.0.0', port = '3303'})
-- raises an error: bad argument options.port to nil (number expected, got string)

configure_connection_opts({login = 'testuser', ip_address = '0.0.0.0', port = 3303})
-- raises an error: unexpected argument options.login to nil
--]]
```

**Примечание:** Table qualifiers can be nested and use tables, too.

### API Reference

| Members | |
|---|---|
| *checks()* | When called inside a function, checks that the function's arguments conform to the specified types |
| *checkers* | A global variable that provides access to checkers for different types |

### checks()

`checks`(*type_1*, ...)
When called inside a function, checks that the function's arguments conform to the specified types.

**Параметры**

- `type_1` (`string/table`) – a *string* or *table* type qualifier used to check the argument type
- `...` – optional type qualifiers used to check the types of *other arguments*

### checkers

The `checkers` global variable provides access to checkers for different types. You can use this variable to add a *custom checker* that performs arbitrary validations.

---

**Примечание:** The `checkers` variable also provides access to checkers for Tarantool-specific types. These checkers can be used in a custom checker.

---

`checkers.datetime`(*value*)
Check whether the specified value is *datetime_ object*.

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is `datetime_object`; otherwise, `false`

**Rtype** boolean

**Example**

```
local datetime = require('datetime')
local is_datetime = checkers.datetime(datetime.new { day = 1, month = 6, year =␣
↪2023 })
local is_interval = checkers.interval(datetime.interval.new { day = 1 })
```

`checkers.decimal`(*value*)
Check whether the specified value has the *decimal* type.

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value has the `decimal` type; otherwise, `false`

**Rtype** boolean

**Example**

```
local decimal = require('decimal')
local is_decimal = checkers.decimal(decimal.new(16))
```

checkers.error(*value*)

Check whether the specified value is *error_ object*.

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is `error_object`; otherwise, `false`

**Rtype** boolean

**Example**

```
local server_error = box.error.new({ code = 500, reason = 'Server error' })
local is_error = checkers.error(server_error)
```

checkers.int64(*value*)

Check whether the specified value is one of the following `int64` values:

- a Lua number in a range from -2^53+1 to 2^53-1 (inclusive)
- Lua cdata `ctype<uint64_t>` in a range from 0 to `LLONG_MAX`
- Lua cdata `ctype<int64_t>`

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is an `int64` value; otherwise, `false`

**Rtype** boolean

**Example**

```
local is_int64 = checkers.int64(-1024)
local is_uint64 = checkers.uint64(2048)
```

checkers.interval(*value*)

Check whether the specified value is *interval_ object*.

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is `interval_object`; otherwise, `false`

**Rtype** boolean

**Example**

```
local datetime = require('datetime')
local is_datetime = checkers.datetime(datetime.new { day = 1, month = 6, year =␣
↪2023 })
local is_interval = checkers.interval(datetime.interval.new { day = 1 })
```

checkers.tuple(*value*)

Check whether the specified value is a *tuple*.

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is a tuple; otherwise, `false`

**Rtype** boolean

**Example**

```
local is_tuple = checkers.tuple(box.tuple.new{1, 'The Beatles', 1960})
```

`checkers.uint64(`*value*`)`

Check whether the specified value is one of the following `uint64` values:

- a Lua number in a range from 0 to 2^53-1 (inclusive)
- Lua cdata `ctype<uint64_t>`
- Lua cdata `ctype<int64_t>` in range from 0 to `LLONG_MAX`

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is an `uint64` value; otherwise, `false`

**Rtype** boolean

**Example**

```
local is_int64 = checkers.int64(-1024)
local is_uint64 = checkers.uint64(2048)
```

`checkers.uuid(`*value*`)`

Check whether the specified value is *uuid_ object*.

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is `uuid_object`; otherwise, `false`

**Rtype** boolean

**Example**

```
local uuid = require('uuid')
local is_uuid = checkers.uuid(uuid())
local is_uuid_bin = checkers.uuid_bin(uuid.bin())
local is_uuid_str = checkers.uuid_str(uuid.str())
```

`checkers.uuid_bin(`*value*`)`

Check whether the specified value is *uuid* represented by a 16-byte binary string.

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is `uuid` represented by a 16-byte binary string; otherwise, `false`

**Rtype** boolean

**See also:** *uuid(value)*

checkers.uuid_str(*value*)
Check whether the specified value is *uuid* represented by a 36-byte hexadecimal string.

**Параметры**

- `value` (`any`) – the value to check the type for

**Return** `true` if the specified value is `uuid` represented by a 36-byte hexadecimal string; otherwise, `false`

**Rtype** boolean

**See also:** *uuid(value)*

### 9.5.4 Модуль clock

#### Общие сведения

Модуль `clock` возвращает значения времени, полученные из функции Posix / C CLOCK_GETTIME или аналогичной. Большинство функций модуля возвращают число секунд; функции, названия которых заканчиваются на «64», возвращают 64-разрядное число наносекунд.

#### Указатель

Ниже приведен перечень всех функций модуля `clock`.

| Имя | Назначение |
|---|---|
| *clock.time() clock.realtime()* | Получение физического времени в секундах |
| *clock.time64() clock.realtime64()* | Получение физического времени в наносекундах |
| *clock.monotonic()* | Получение монотонного времени в секундах |
| *clock.monotonic64()* | Получение монотонного времени в наносекундах |
| *clock.proc()* | Получение времени процессора в секундах |
| *clock.proc64()* | Получение времени процессора в наносекундах |
| *clock.thread()* | Получение рабочего времени потока в секундах |
| *clock.thread64()* | Получение рабочего времени потока в наносекундах |
| *clock.bench()* | Измерение времени, которое функция проводит в процессоре |

clock.time()
clock.time64()
clock.realtime()
clock.realtime64()
Физическое время в секундах. Получено из C-функции `clock_gettime(CLOCK_REALTIME)`.

**возвращает** секунды или наносекунды с начала отсчета (1970-01-01 00:00:00), значение корректируется.

**тип возвращаемого значения** number or cdata (ctype<int64_t>)

**Пример:**

```
-- Результатом будет примерное число лет с 1970.
clock = require('clock')
print(clock.time() / (365*24*60*60))
```

См. также *fiber.time64* и стандартную Lua-функцию os.clock.

clock.monotonic()

clock.monotonic64()
The monotonic time. Derived from C function clock_gettime(CLOCK_MONOTONIC). Monotonic time is similar to wall clock time but is not affected by changes to or from daylight saving time, or by changes done by a user. This is the best function to use with benchmarks that need to calculate elapsed time.

**возвращает** секунды или наносекунды с момента последней загрузки компьютера.

**тип возвращаемого значения** number or cdata (ctype<int64_t>)

**Пример:**

```
-- Результатом будет число наносекунд с запуска.
clock = require('clock')
print(clock.monotonic64())
```

clock.proc()
clock.proc64()
Время процессора. Получено из C-функции clock_gettime(CLOCK_PROCESS_CPUTIME_ID). Такую функцию лучше всего использовать для эталонного тестирования, где необходимо рассчитать время, затраченное на процессоре.

**возвращает** секунды или наносекунды с момента начала работы процессора.

**тип возвращаемого значения** number or cdata (ctype<int64_t>)

**Пример:**

```
-- Результатом будет число наносекунд с запуска процессора.
clock = require('clock')
print(clock.proc64())
```

clock.thread()
clock.thread64()
Рабочее время потока. Получено из C-функции clock_gettime(CLOCK_THREAD_CPUTIME_ID). Такую функцию лучше всего использовать для эталонного тестирования, где необходимо рассчитать время, затраченное потоком на процессоре.

**возвращает** секунды или наносекунды с момента начала работы потока процессора транзакций.

**тип возвращаемого значения** number or cdata (ctype<int64_t>)

**Пример:**

```
-- Результатом будет число секунд с момента начала работы потока.
clock = require('clock')
print(clock.thread64())
```

clock.bench(*function*[, ...])
Время, которое функция проводит в процессоре. Данная функция использует clock.proc(), то есть рассчитывает затраченное процессором время. Таким образом, она не используется для отображения фактически затраченного времени.

**Параметры**

- function (function) – функция или ссылка на функцию
- ... – значения, которые необходимы для функции.

**возвращает таблица**. Первый элемент – время работы процессора в секундах, второй элемент – то, что возвращает функция.

**Пример:**

```
-- Эталонное тестирование функции, которая находится в спящем режиме в течение 10␣
↪секунд.
-- NB: bench() не будет рассчитывать время сна.
-- Поэтому вернется значение, которое будет {число менее 10, 88}.
clock = require('clock')
fiber = require('fiber')
function f(param)
  fiber.sleep(param)
  return 88
end
clock.bench(f, 10)
```

### 9.5.5 Module compat

Module `compat` is introduced since version 2.11.0-rc.

The usual way to handle compatibility problems is to introduce an option for a new behavior and leave the old one by default. It is not always the perfect way.

Sometimes developers want to keep the old behavior for existing applications and offer the new behavior by default for the new ones. For example, the old behavior is known to be problematic, or less safe, or it doesn't correspond to user expectations. In contrast, the user doesn't always read all the documentation and often assumes good defaults. It was decided to introduce a compatibility module to provide a direct way to deprecate unwanted behavior.

The `compat` module is basically a global table of options with additional verbose interface and helper functions. There are three stages of changing behavior:

1. Old behavior by default.
2. New behavior by default.
3. New behavior is frozen and the old behavior is removed.

During the first two stages, a user can toggle options via the interface and change the behavior according to one's needs. At the last stage, the old behavior is removed from the codebase, and the option is marked as obsolete. Because `compat` is a global instance, options can be hardcoded into it or added in runtime, for example, by external module.

Options are switched to the next stage in major releases. In this way, developers are able to adapt to the new standard behavior and test it before switching to the next release. If something is broken by a new Tarantool version, a developer can still have a way to fix it by a simple config change, that is, explicitly select the old behavior.

Consider example below:

- The option `json_esc_slash` is introduced in the 2.11 minor release. Default is set to „old“, but a developer can utilize the new behavior or test the updated behavior by switching it manually to „new“.
- In release 3.0, the next major release, `json_esc_slash` default is switched to „new“. Now, developers who don't manage to adapt to the new behavior, are able to switch the option to „old“ and fix their module in the future.
- In release 4.0, `json_esc_slash` is marked as obsolete, and the old behavior is no longer accessible. Developers are forced to use the new behavior.

### Basic usage

If you want to explicitly secure every behavior in `compat`, you can do it manually, and then call `compat.dump()` to get a Lua command that sets up the `compat` with all the options selected. You should place this commands at the beginning of code in your `init.lua` file. In this way, you are guaranteed to get the same behavior on any other Tarantool version. See a *tutorial on using compat* for more examples.

### Options

Below are the available `compat` options:

- *json_escape_forward_slash*
- *yaml_pretty_multiline*
- *fiber_channel_close_mode*
- *box_cfg_replication_sync_timeout*
- *sql_seq_scan_default*
- *fiber_slice_default*

### JSON encode escape forward slash

For some reason, in the upstream `lua_cjson`, the „/“ sign is escaped. But according to the `rfc4627` standard, it is unnecessary and questionably compatible with other implementations.

### Old and new behavior

By toggling the `json_escape_forward_slash` compat option, you can chose either the json encoder escapes the „/“ sign or it does not:

```
tarantool> require('compat').json_escape_forward_slash = 'old'
---
...

tarantool> require('json').encode('foo/bar')
---
- '"foo\/bar"'
...

tarantool> require('compat').json_escape_forward_slash = 'new'
---
...

tarantool> require('json').encode('foo/bar')
---
- '"foo/bar"'
...
```

The option affects both the global serializer instance and serializers created with `json.new()`. It also affects the way log messages are encoded when written to the log in the json format (the `box.cfg.log_format` option is set to „json“).

#### Known compatibility issues

At this point, no incompatible modules are known.

#### Detecting issues in your codebase

Both encoding styles are correct from the JSON standard standpoint, but if your module relies on encodings results bytewise, it may break with this change. Be cautious if you do the following:

- Hash results of `json.encode()`.

### Lua-YAML prettier multiline output

The `lua-yaml` encoder selects the string style automatically, but in Tarantool context, it can be beneficial to enforce them, for example, for better readability. The `yaml_pretty_multiline` compat option allows to encode multiline strings in a block style.

#### Old and new behavior

The `compat` module allows you to chose between the `lua-yaml` encodes multiline strings as usual or in the enforced block scalar style:

```
tarantool> compat.yaml_pretty_multiline = 'old'
---
...

tarantool> return "Title: xxx\n- Item 1\n- Item 2\n"
---
- 'Title: xxx

  - Item 1

  - Item 2

  '
...

tarantool> compat.yaml_pretty_multiline = 'new'
---
...

tarantool> return "Title: xxx\n- Item 1\n- Item 2\n"
---
- |
  Title: xxx
  - Item 1
  - Item 2
...
```

You can select the new/old behavior in `compat`. It affects the global YAML encoder.

#### Known compatibility issues

At this point, no incompatible modules are known.

#### Detecting issues in your codebase

Both encoding styles are correct from the YAML standard standpoint, but if your module relies on encodings results bytewise, it may break with this change. Be cautious if you do the following:

- Compare results of YAML encoding as strings.
- Hash results of yaml encoding.

### Fiber channel close mode

Before the change, there was an unexpected behavior when using `channel:close()` because it closed the channel entirely and discarded all unread events.

#### Old and new behavior

The `compat` module allows you chose between the channel force and graceful closing. The latter is a new behavior.

```
tarantool> compat = require('compat')
---
...

tarantool> compat
---
- - yaml_pretty_multiline: default (old)
  - json_escape_forward_slash: default (old)
  - fiber_channel_close_mode: default (old)
...

tarantool> fiber = require('fiber')
---
...

tarantool> ch = fiber.channel(10)
---
...

tarantool> ch:put('one')
---
- true
...

tarantool> ch:put('two')
---
- true
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> ch:get()
---
- one
...

tarantool> ch:close()
---
...

tarantool> ch:get()
---
- null
...

tarantool> compat.fiber_channel_close_mode = 'new'
---
...

tarantool> ch = fiber.channel(10)
---
...

tarantool> ch:put('one')
---
- true
...

tarantool> ch:put('two')
---
- true
...

tarantool> ch:get()
---
- one
...

tarantool> ch:close()
---
...

tarantool> ch:get()
---
- two
...

tarantool> ch:get()
---
- null
...
```

You can select new/old behavior in `compat`. It will affect all existing channels and the future ones.

### Known compatibility issues

At this point, no incompatible modules are known.

### Detecting issues in your codebase

The new behavior is mostly backward compatible. The only known problem that can appear is when the code relies on channel being entirely closed after `ch:close()` and `ch:get()` returning `nil`.

### Default value for replication_sync_timeout

Having a non-zero *replication_sync_timeout* gives a user the false assumption that the `box.cfg{replication = ...}` call returns only when the configured node is synced with all the other nodes. This is mostly true for the big `replication_sync_timeout` values, but it is not 100% guaranteed. In other words, a user still has to check if the node is synced, or the sync just timed out. Besides, while `replication_sync_timeout` is ticking, you cannot reconfigure `box` with another `box.cfg` call, which hardens reconfiguration.

It is decided to set the `replication_sync_timeout` to zero by default.

### Old and new behavior

The `compat` module allows you to choose between

- the old behavior: `box.cfg.replication_sync_timeout` is 300 seconds by default
- and the new behavior:`box.cfg.replication_sync_timeout` is 0 by default.

It is important to set the desired behavior before the initial `box.cfg{}` call to take effect for it.

```
tarantool> compat.box_cfg_replication_sync_timeout = 'new'
---
...
tarantool> box.cfg{}
---
...
tarantool> box.cfg.replication_sync_timeout
---
- 0
...
tarantool> compat.box_cfg_replication_sync_timeout = 'old'
---
- error: 'builtin/box/load_cfg.lua:253: The compat  option ''box_cfg_replication_sync_
↪timeout''
    takes effect only before the initial box.cfg() call'
...
```

A fresh Tarantool run:

```
tarantool> compat.box_cfg_replication_sync_timeout = 'old'
---
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> box.cfg{}
---
...
tarantool> box.cfg.replication_sync_timeout
---
- 300
...
```

### Known compatibility issues

At this point, no incompatible modules are known.

### Detecting issues in your codebase

We expect issues with a user assuming that the node is not in the orphan state (`box.info.status ~= "orphan"`) after the `box.cfg{replication=...}` call returns. This is not true with the new behaviour. To simulate the old behavior, one may add a `box.ctl.wait_rw()` call after the `box.cfg{}` call. `box.ctl.wait_rw()` returns only when the node becomes writable, and hence is not an orphan.

### Default value for sql_seq_scan session setting

The default value for the `sql_seq_scan` session setting will be set to false starting with Tarantool 3.0. To be able to return the behavior to the old default, a new `compat` option is introduced.

### Old and new behavior

Old behavior: SELECT scan queries are always allowed.

New behavior: SELECT scan queries are only allowed if the SEQSCAN keyword is used correctly.

Note that the `sql_seq_scan_default` compat option only affects sessions during initialization. It means that you should set `sql_seq_scan_default` before running `box.cfg{}` or creating a new session. Also, a new session created before executing `box.cfg{}` will not be affected by the value of the `compat` option.

Examples of setting the option before execution of `box.cfg{}`:

```
tarantool> require('compat').sql_seq_scan_default = 'new'
---
...

tarantool> box.cfg{log_level = 1}
---
...

tarantool> box.space._session_settings:get{'sql_seq_scan'}
---
- ['sql_seq_scan', false]
...
```

```
tarantool> require('compat').sql_seq_scan_default = 'old'
---
...

tarantool> box.cfg{log_level = 1}
---
...

tarantool> box.space._session_settings:get{'sql_seq_scan'}
---
- ['sql_seq_scan', true]
...
```

Examples of setting the option before creation of a new session after execution of `box.cfg{}`:

```
tarantool> box.cfg{log_level = 1, listen = 3301}
---
...

tarantool> require('compat').sql_seq_scan_default = 'new'
---
...

tarantool> cn = require('net.box').connect(box.cfg.listen)
---
...

tarantool> cn.space._session_settings:get{'sql_seq_scan'}
---
- ['sql_seq_scan', false]
...

tarantool> require('tarantool').compat.sql_seq_scan_default = 'old'
---
...

tarantool> cn = require('net.box').connect(box.cfg.listen)
---
...

tarantool> cn.space._session_settings:get{'sql_seq_scan'}
---
- ['sql_seq_scan', true]
...
```

#### Known compatibility issues

At this point, no incompatible modules are known.

#### Detecting issues in your codebase

We expect most SELECTs that do not use indexes to fail after the `sql_seq_scan` session setting is set to false. The best way to avoid this is to refactor the query to use indexes. To understand if SELECT uses indexes, you can use EXPLAIN QUERY PLAN. If SEARCH TABLE is specified, the index is used. If it says SCAN TABLE, the index is not used.

You can use the SEQSCAN keyword to manually allow scanning queries. Or you can set the `sql_seq_scan` session setting to true to allow all scanning queries.

### Default value for max fiber slice

The max fiber slice specifies the max fiber execution time without yield before a warning is logged or an error is raised. It is set with the *fiber.set_max_slice()* function. The new `compat` option – `fiber_slice_default` – controls the default value of the max fiber slice.

#### Old and new behavior

The old default value for the max fiber slice is infinity (no warnings or errors). The new default value is `{warn = 0.5, err = 1.0}`. To use the new behavior, set `fiber_slice_default` to `new` as follows:

```
compat = require('compat')
compat.fiber_slice_default = 'new'
```

#### Known compatibility issues

At this point, no incompatible modules are known.

#### Detecting issues in your codebase

If you see a warning like this in the log:

`fiber has not yielded for more than 0.500 seconds`,

or the following error is raised unexpectedly by a `box` function

`error: fiber slice is exceeded`,

then your application has a fiber that may exceed its slice and fail.

First, make sure that *fiber.yield()* is used for this fiber to transfer control to another fiber. You can also extend the fiber slice with the *fiber.extend_slice(slice)* function.

### Tutorial: Module compat

This tutorial covers the following `compat` module API and its usage:


- *Listing options*
- *Listing options details*
- *Changing option value*
- *Restoring defaults*
- *Getting compat setup with compat.dump()*
- *Setting all options to a specific value with compat.dump()*
- *Adding an option during runtime*


### Listing options

The options list is serialized in the interactive console with additional details for user convenience:

- All non-obsolete options in order new > old > default.
- Serialization returns array-like table with tables `{<option> = <value>}`.
- The result of compat serialization can still be indexed as a normal key-value table.

```
tarantool> compat = require('compat')
---
...

tarantool> compat
---
- - json_escape_forward_slash: new
- - option_2: old
- - option_default_old: default (old)
- - option_default_new: default (new)
...
```

### Listing options details

- `current` – the state of the option.
- `default` – the default state of the option.
- `brief` – text options description with a link to more detailed description.

```
tarantool> compat.option_default_new
---
- current: old
default: new
brief: <...>
...
```

### Changing option value

You can do it directly, or by passing a table with option-value. Possible values to assign are „new“ , „old“, and „default“.

```
tarantool> compat.json_escape_forward_slash = 'old'
---
...

tarantool> compat{json_escape_forward_slash = 'new', option_2 = 'default'}
---
...
```

### Restoring defaults

By setting „default“ value to an option:

```
tarantool> compat.option_2 = 'default'
---
...

tarantool> compat.option_2
---
- current: default
- default: new
- brief: <...>
...
```

### Getting compat setup with compat.dump()

```
tarantool> compat({
         >     obsolete_set_explicitly = 'new',
         >     option_set_old = 'old',
         >     option_set_new = 'new'
         > })
---
...

tarantool> compat
---
- - option_set_old: old
- - option_set_new: new
- - option_default_old: default (old)
- - option_default_new: default (new)
...

# Obsolete options are not returned in serialization, but have the following values:
# - obsolete_option_default: default (new)
# - obsolete_set_explicitly: new
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# nil does output obsolete unset options as 'default'
tarantool> compat.dump()
---
- require('compat')({
            option_set_old          = 'old',
            option_set_new          = 'new',
            option_default_old      = 'default',
            option_default_new      = 'default',
            obsolete_option_default = 'default', -- obsolete since X.Y
            obsolete_set_explicitly = 'new',     -- obsolete since X.Y
    })
...

# 'current' is the same as nil with default set to current values
tarantool> compat.dump('current')
---
- require('compat')({
            option_set_old          = 'old',
            option_set_new          = 'new',
            option_default_old      = 'old',
            option_default_new      = 'new',
            obsolete_option_default = 'new',     -- obsolete since X.Y
            obsolete_set_explicitly = 'new',     -- obsolete since X.Y
    })
...

# 'new' outputs obsolete as 'new'.
tarantool> compat.dump('new')
---
- require('compat')({
            option_set_old          = 'new',
            option_set_new          = 'new',
            option_default_old      = 'new',
            option_default_new      = 'new',
            obsolete_option_default = 'new',     -- obsolete since X.Y
            obsolete_set_explicitly = 'new',     -- obsolete since X.Y
    })
...

# 'old' outputs obsolete options as 'new'.
tarantool> compat.dump('old')
---
- require('compat')({
            option_set_old          = 'old',
            option_set_new          = 'old',
            option_default_old      = 'old',
            option_default_new      = 'old',
            obsolete_option_default = 'new',     -- obsolete since X.Y
            obsolete_set_explicitly = 'new',     -- obsolete since X.Y
    })
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# 'default' does output obsolete options as default.
tarantool> dump('default')
---
- require('compat')({
            option_set_old          = 'default',
            option_set_new          = 'default',
            option_default_old      = 'default',
            option_default_new      = 'default',
            obsolete_option_default = 'default', -- obsoleted since X.Y
            obsolete_set_explicitly = 'default', -- obsoleted since X.Y
    })
...
```

### Setting all options to a specific value with compat.dump()

- use compat.dump() to get a specific configuration
- copy and paste it into console (or use loadstring())

```
tarantool> compat.dump('new')
---
- require('compat')({
      option_2 = 'new',
      json_escape_forward_slash = 'new',
  })
...
tarantool> require('compat')({
      option_2 = 'new',
      json_escape_forward_slash = 'new',
  })
---
...

tarantool> compat
---
- - json_escape_forward_slash: new
- - option_2: new
...
```

### Adding an option during runtime

User must provide a table with:

- name (string)
- default ('new' / 'old')
- brief (explanation of the option, can be multiline string)
- obsolete ('X.Y' / nil) — tarantool version that marked option as obsolete. When nil, option is treated as non-obsolete)
- action function (argument - boolean is_new, changes the behavior accordingly)

- run_action_now (true / false / nil) if add_options should run action afterwards, false by default

Option hot reload:

You can change an existing option in runtime using add_option(), it will update all the fields but keep currently selected behavior if any. The new action will be called afterwards.

```
tarantool> compat.add_option{
                 name = 'option_4',
                 default = 'new',
                 brief = "<...>",
                 obsolete = nil,          -- you can explicitly mark the option as non-
↪obsolete
                 action = function(is_new)
                      print(("option_4 action was called with is_new = %s!"):format(is_
↪new))
                 end,
                 run_action_now = true
           }
option_4 postaction was called with is_new = true!
---
...

tarantool> compat.add_option{              -- hot reload of option_4
                 name = 'option_4',
                 default = 'old',          -- different default
                 brief = "<...>",
                 action = function(is_new)
                      print(("new option_4 action was called with is_new = %s!
↪"):format(is_new))
                 end
           }
---
...         -- action is not called by default
```

### 9.5.6 Module config

**Since:** *3.0.0*

The `config` module provides the ability to work with an instance's configuration. For example, you can determine whether the current instance is up and running without errors after applying the *cluster's configuration*.

By using the `config.storage` *role*, you can set up a Tarantool-based *centralized configuration storage* and interact with this storage using the `config` module API.

### Loading config

To load the `config` module, use the `require()` directive:

```
local config = require('config')
```

Then, you can access its *API*:

```
config:reload()
```

### API Reference

| **config API** | |
|---|---|
| *config:get()* | Get a configuration applied to the current instance |
| *config:info()* | Get the current instance's state in regard to configuration |
| *config:reload()* | Reload the current instance's configuration |
| **config.storage API** | |
| *config.storage.put()* | Put a value by the specified path |
| *config.storage.get()* | Get a value stored by the specified path |
| *config.storage.delete()* | Delete a value stored by the specified path |
| *config.storage.info()* | Get information about an instance's connection state |
| *config.storage.txn()* | Make an atomic request |

### config API

`object config`

`config:get([`*param*`])`

Get a configuration applied to the current instance. Optionally, you can pass a configuration option name to get its value.

**Параметры**

- `param` (*string*) – a configuration option name

**Return** an instance configuration

**Examples:**

The example below shows how to get the full instance configuration:

```
app:instance001> require('config'):get()
---
- fiber:
    io_collect_interval: null
    too_long_threshold: 0.5
    top:
      enabled: false
  # other configuration values
  # ...
```

This example shows how to get an `iproto.listen` option value:

```
app:instance001> require('config'):get('iproto.listen')
---
- - uri: 127.0.0.1:3301
...
```

`config.get()` can also be used in *application code* to get the value of a custom configuration option.

`config:info()`

Get the current instance's state in regard to configuration.

**Return** a table containing an instance's state

The returned state includes the following sections:

- `status` – one of the following statuses:
  - `ready` – the configuration is applied successfully
  - `check_warnings` – the configuration is applied with warnings
  - `check_errors` – the configuration cannot be applied due to configuration errors
- `meta` – additional configuration information
- `alerts` – warnings or errors raised on an attempt to apply the configuration

**Examples:**

Below are a few examples demonstrating how the `info()` output might look:

- In the example below, an instance's state is `ready` and no warnings are shown:

```
app:instance001> require('config'):info()
---
- status: ready
  meta: []
  alerts: []
...
```

- In the example below, the instance's state is `check_warnings`. The `alerts` section informs that privileges to the `books` space for `sampleuser` cannot be granted because the `books` space has not created yet:

```
app:instance001> require('config'):info()
---
- status: check_warnings
  meta: []
  alerts:
  - type: warn
    message: box.schema.user.grant("sampleuser", "read,write", "space",
↪"books") has
      failed because either the object has not been created yet, or the␣
↪privilege
      write has failed (separate alert reported)
    timestamp: 2024-02-27T15:07:41.815785+0300
...
```

This warning is cleared when the `books` space is created.

- In the example below, the instance's state is `check_errors`. The `alerts` section informs that the `log.level` configuration option has an incorrect value:

```
app:instance001> require('config'):info()
---
- status: check_errors
  meta: []
  alerts:
  - type: error
    message: '[cluster_config] log.level: Got 8, but only the following␣
↪values are
      allowed: 0, fatal, 1, syserror, 2, error, 3, crit, 4, warn, 5, info,␣
↪6, verbose,
      7, debug'
    timestamp: 2024-02-29T12:55:54.366810+0300
...
```

- In this example, an instance's state includes information about a *centralized storage* the instance takes a configuration from:

```
app:instance001> require('config'):info()
---
- status: ready
  meta:
    storage:
      revision: 8
      mod_revision:
        /myapp/config/all: 8
  alerts: []
...
```

`config:reload()`
Reload the current instance's configuration. Below are a few use cases when this function can be used:

- A configuration option value specific to this instance is changed in a cluster's configuration.
- A new instance is *added to a replica set*.
- A centralized configuration with turned-off configuration reloading is updated. Learn more at *Reloading configuration*.

### config.storage API

The `config.storage` API allows you to interact with a Tarantool-based *centralized configuration storage*.

`config.storage.put(`*path*, *value*`)`
Put a value by the specified path.

**Параметры**

- `path` (*string*) – a path to put the value by
- `value` (*string*) – a value to put

**Return** a table containing the following fields:

- `revision`: a revision after performing the operation

**Rtype** *table*

**Example:**

The example below shows how to read a configuration stored in the `source.yaml` file using the *fio module* API and put this configuration by the `/myapp/config/all` path:

```
local fio = require('fio')
local cluster_config_handle = fio.open('../../source.yaml')
local cluster_config = cluster_config_handle:read()
local response = config.storage.put('/myapp/config/all', cluster_config)
cluster_config_handle:close()
```

Example on GitHub: tarantool_config_storage.

`config.storage.get(`*path*`)`

Get a value stored by the specified path or prefix.

**Параметры**

- `path` (*string*) – a path or prefix to get a value by; prefixes end with `/`

**Return** a table containing the following fields:

- `data`: a table containing the information about the value:
  - `path`: a path
  - `mod_revision`: a last revision at which this value was modified
  - `value:`: a value
- `revision`: a revision after performing the operation

**Rtype** *table*

**Examples:**

The example below shows how to get a configuration stored by the `/myapp/config/all` path:

```
local response = config.storage.get('/myapp/config/all')
```

This example shows how to get all configurations stored by the `/myapp/` prefix:

```
local response = config.storage.get('/myapp/')
```

Example on GitHub: tarantool_config_storage.

`config.storage.delete(`*path*`)`

Delete a value stored by the specified path or prefix.

**Параметры**

- `path` (*string*) – a path or prefix to delete the value by; prefixes end with `/`

**Return** a table containing the following fields:

- `data`: a table containing the information about the value:
  - `path`: a path
  - `mod_revision`: a last revision at which this value was modified
  - `value:`: a value
- `revision`: a revision after performing the operation

**Rtype** *table*

**Examples:**

The example below shows how to delete a configuration stored by the `/myapp/config/all` path:

```
local response = config.storage.delete('/myapp/config/all')
```

In this example, all configuration are deleted:

```
local response = config.storage.delete('/')
```

Example on GitHub: tarantool_config_storage.

`config.storage.info()`
Get information about an instance's connection state.

**Return** a table containing the following fields:

- `status`: a connection status, which can be one of the following:
  - `connected`: if any instance from the quorum is available to the current instance
  - `disconnected`: if the current instance doesn't have a connection with the quorum

**Rtype** *table*

`config.storage.txn(`*request*`)`
Make an atomic request.

**Параметры**

- `request` (*table*) – a table containing the following optional fields:
  - `predicates`: a list of predicates to check. Each predicate is a list that contains:

    ```
    {target, operator, value[, path]}
    ```

    * `target` – one of the following string values: `revision`, `mod_revision`, `value`, `count`
    * `operator` – a string value: `eq`, `ne`, `gt`, `lt`, `ge`, `le` or its symbolic equivalent, for example, `==`, `!=`, `>`
    * `value` – an unsigned or string value to compare
    * `path` (optional) – a string value: can be a path with the `mod_revision` and `value` target or a path/prefix with the `count` target
  - `on_success`: a list with operations to execute if all predicates in the list evaluate to `true`
  - `on_failure`: a list with operations to execute if any of a predicate evaluates to `false`

**Return** a table containing the following fields:

- `data`: a table containing response data:
  - `responses`: the list of responses for all operations
  - `is_success`: a boolean value indicating whether the predicate is evaluated to `true`
- `revision`: a revision after performing the operation

**Rtype** *table*

**Example:**

```
local response = config.storage.txn({
    predicates = { { 'value', '==', 'v0', '/myapp/config/all' } },
    on_success = { { 'put', '/myapp/config/all', 'v1' } },
    on_failure = { { 'get', '/myapp/config/all' } }
})
```

Example on GitHub: tarantool_config_storage.

### 9.5.7 Модуль console

#### Общие сведения

Модуль `console` позволяет одному экземпляру Tarantool получать доступ к другому экземпляру Tarantool и позволяет одному экземпляру Tarantool начать прослушивание по *порту администрирования*.

#### Указатель

Ниже приведен перечень всех функций модуля `console`.

| Имя | Назначение |
|---|---|
| *console.connect()* | Подключение к экземпляру |
| *console.listen()* | Прослушивание входящих запросов |
| *console.start()* | Запуск консоли |
| *console.ac()* | Установка флага автодополнения ввода |
| *console.delimiter()* | Настройка разделителя |
| *console.get_default_output()* | Get default output format |
| *console.set_default_output()* | Set default output format |
| *console.eos()* | Set or get end-of-output string |

`console.connect(`*uri*`)`

Подключение к экземпляру по *URI*, смена командной строки с „`tarantool>`“ на „*uri* >“ и дальнейшая работа в качестве клиента до окончания сессии пользователя или ввода команды `control-D`.

Функция console.connect позволяет одному экземпляру Tarantool в интерактивном режиме получать доступ к другому экземпляру Tarantool. Последующие запросы на первый взгляд будут обрабатываться локально, но в действительности запросы отправляются на удаленный экземпляр, а локальный экземпляр выступает в виде клиента. После успешного подключения командная строка сменится, и последующие запросы отправляются и выполняются на удаленном экземпляре. Результат выводится на локальный экземпляр. Чтобы вернуться к работе на локальном экземпляре, введите команду `control-D`.

Если экземпляр Tarantool по `URI` запрашивает авторизацию, подключение может выглядеть следующим образом: `console.connect('admin:secretpassword@distanthost.com:3301')`.

Нет ограничений по типу вводимых запросов, кроме ограничений по правам на выполняемые запросы – по умолчанию, вход в систему на удаленном экземпляре выполняется от имени пользователя „guest“. Можно разрешить работу на удаленном экземпляре, выдав права: `box.schema.user.grant('guest','execute','universe')`.

**Параметры**

- `uri` (*string*) – URI удаленного экземпляра

**возвращает** nil

Возможные ошибки: подключение не будет установлено, если целевой экземпляр Tarantool не был инициирован с помощью `box.cfg{listen=...}`.

**Пример:**

```
tarantool> console = require('console')
---
...
tarantool> console.connect('198.18.44.44:3301')
---
...
198.18.44.44:3301> -- командная строка показывает, что работа идет с удаленным
 ↪экземпляром
```

`console.listen(uri)`

Прослушивание по *URI*. Основной способ прослушивания на предмет входящих запросов – по строке информации о подключении, или URI, указанному в `box.cfg{listen=...}`. Другой способ прослушивания – по URI, указанному в `console.listen(...)`. Этот другой способ называется «административным» или просто *«по порту администрирования»*. Такое прослушивание обычно осуществляется по локальному хосту с доменным Unix-сокетом.

**Параметры**

- `uri` (*string*) – URI локального экземпляра

«Административный» адрес – это URI для прослушивания. У него нет значения по умолчанию, поэтому следует указать, будет ли подключение производиться по порту администрирования. Параметр выражен URI = Универсальным идентификатором ресурса, например «/tmpdir/unix_domain_socket.sock», или числовым идентификатором TCP-порта. Подключения часто выполняются по telnet. Типичное значение порта: 3313.

**Пример:**

```
tarantool> console = require('console')
---
...
tarantool> console.listen('unix/:/tmp/X.sock')
... main/103/console/unix/:/tmp/X I> started
---
- fd: 6
  name:
    host: unix/
    family: AF_UNIX
    type: SOCK_STREAM
    protocol: 0
    port: /tmp/X.sock
...
```

`console.start()`

Запуск консоли на текущем интерактивном терминале.

**Пример:**

`console.start()` специально используется с *файлами инициализации*. Как правило, при запуске экземпляра Tarantool с помощью команды `tarantool` *`initialization file`*, консоль не поддерживается. Эту проблему можно решить путем добавления следующих строк в конце файла инициализации:

```
local console = require('console')
console.start()
```

`console.ac(`[*true/false*]`)`

Установка флага автодополнения ввода. Если значение автодополнения = `true` (правда), и пользователь использует Tarantool в качестве клиента или подключен к Tarantool по `console.connect()`, то при нажатии клавиши TAB Tarantool будет автоматически дополнять текст по введенной части. По умолчанию, задано значение `true`.

`console.delimiter(`*marker*`)`

Настройка специального маркера окончания запроса для консоли Tarantool.

По умолчанию, маркер окончания запроса представляет собой символ разрыва строки (перевод строки). Нет необходимости в специальных маркерах, поскольку Tarantool может определить, если многостроковый запрос не завершен (например, если видно, что при объявлении функции еще не задано конечное ключевое слово). Тем не менее, в особых случаях или при вводе многостроковых запросов в более ранних версиях Tarantool, можно изменить маркер окончания запроса. В результате символ разрыва строки не будет означать окончание запроса.

Чтобы вернуться в нормальный режим, введите команду: `console.delimiter('')<marker>`

**Параметры**

- `marker` (*string*) – специальный маркер окончания запроса для консоли Tarantool

**Пример:**

```
tarantool> console = require('console'); console.delimiter('!')
---
...
tarantool> function f ()
         > statement_1 = 'a'
         > statement_2 = 'b'
         > end!
---
...
tarantool> console.delimiter('')!
---
...
```

`console.get_default_output()`

Return the current default output format. The result will be `fmt="yaml"`, or it will be `fmt="lua"` if the last *set_default_output* call was `console.set_default_output('lua')`.

`console.set_default_output(`*'yaml'/'lua'*`)`

Set the default output format. The possible values are „yaml“ (the default default) or „lua“. The output format can be changed within a session by executing `console.eval('\set output` *`yaml`* | *`lua`*`')`; see the description of output format in the *Interactive console* section.

`console.eos(`[*string*]`)`

Set or access the end-of-output string if default output is „lua“. This is the string that appears at the end of output in a response to any Lua request. The default value is `;` semicolon. Saying `eos()` will

return the current value. For example, after `require('console').eos('!!')` responses will end with „!!“.

### 9.5.8 Модуль crypto

#### Общие сведения

«Crypto» is short for «Cryptography», which generally refers to the production of a digest value from a function (usually a Cryptographic hash function), applied against a string. Tarantool's `crypto` module supports ten types of cryptographic hash functions (AES, DES, DSS, MD4, MD5, MDC2, RIPEMD, SHA-1, SHA-2). Some of the crypto functionality is also present in the *Модуль digest* module.

#### Указатель

Ниже приведен перечень всех функций модуля `crypto`.

| Имя | Назначение |
|---|---|
| *crypto.cipher.{algorithm}.{cipher_mode}.encrypt()* | Шифрование строки |
| *crypto.cipher.{algorithm}.{cipher_mode}.decrypt()* | Расшифрование строки |
| *crypto.digest.{algorithm}()* | Получение дайджеста |
| *crypto.hmac.{algorithm}()* | Получение хеш-ключа |

`crypto.cipher.{aes128|aes192|aes256|des}.{cbc|cfb|ecb|ofb}.encrypt`(*string*, *key*, *initialization_vector*)

`crypto.cipher.{aes128|aes192|aes256|des}.{cbc|cfb|ecb|ofb}.decrypt`(*string*, *key*, *initialization_vector*)

Передача или возврат шифрованного сообщения, полученного из строки, ключа и (необязательно) вектора инициализации. Четыре алгоритма на выбор:

- aes128 - aes-128 (128-битные двоичные строки с использованием AES)
- aes192 - aes-192 (192-битные двоичные строки с использованием AES)
- aes256 - aes-256 (256-битные двоичные строки с использованием AES)
- des - des (56-битные двоичные строки с использованием DES, хотя использование DES не рекомендуется)

Также доступны четыре режима блочного шифрования на выбор:

- cbc - Сцепление блоков шифротекста
- cfb - Обратная связь по шифротексту
- ecb - Электронная кодовая книга
- ofb - Обратная связь по выходу

For more information, read the article about Encryption Modes

**Пример:**

```
_16byte_iv='1234567890123456'
_16byte_pass='1234567890123456'
e=crypto.cipher.aes128.cbc.encrypt('string', _16byte_pass, _16byte_iv)
crypto.cipher.aes128.cbc.decrypt(e,  _16byte_pass, _16byte_iv)
```

`crypto.digest.{dss|dss1|md4|md5|mdc2|ripemd160}`(*string*)

crypto.digest.{sha1|sha224|sha256|sha384|sha512}(*string*)

Передача или возврат дайджеста из строки. Выбор из одиннадцати алгоритмов:

- dss - dss (с использованием DSS)
- dss1 - dss (с использованием DSS-1)
- md4 - md4 (128-битные двоичные строки с использованием MD4)
- md5 - md5 (128-битные двоичные строки с использованием MD5)
- mdc2 - mdc2 (с использованием MDC2)
- ripemd160 - ripemd (160-битные двоичные строки с использованием RIPEMD-160)
- sha1 - sha-1 (160-битные двоичные строки с использованием SHA-1)
- sha224 - sha-224 (224-битные двоичные строки с использованием SHA-2)
- sha256 - sha-256 (256-битные двоичные строки с использованием SHA-2)
- sha384 - sha-384 (384-битные двоичные строки с использованием SHA-2)
- sha512 - sha-512(512-битные двоичные строки с использованием SHA-2).

**Пример:**

```
crypto.digest.md4('string')
crypto.digest.sha512('string')
```

crypto.hmac.{md4|md5|ripemd160}(*key*, *string*)
crypto.hmac.{sha1|sha224|sha256|sha384|sha512}(*key*, *string*)

Передача ключа и строки. Результатом будет код аутентификации сообщения HMAC. 8 алгоритмов на выбор:

- md4 или md4_hex - md4 (128-битные двоичные строки с использованием MD4)
- md5 или md5_hex - md5 (128-битные двоичные строки с использованием MD5)
- ripemd160 или ripemd160_hex - ripemd (160-битные двоичные строки с использованием RIPEMD-160)
- sha1 или sha1_hex - sha-1 (160-битные двоичные строки с использованием SHA-1)
- sha224 или sha224_hex - sha-224 (224-битные двоичные строки с использованием SHA-2)
- sha256 или sha256_hex - sha-256 (256-битные двоичные строки с использованием SHA-2)
- sha384 или sha384_hex - sha-384 (384-битные двоичные строки с использованием SHA-2)
- sha512 или sha512_hex - sha-512(512-битные двоичные строки с использованием SHA-2).

**Пример:**

```
crypto.hmac.md4('key', 'string')
crypto.hmac.md4_hex('key', 'string')
```

### Инкрементальные методы в модуле crypto

Предположим, что вычислен дайджест для строки „A“, затем часть „B“ добавляется в строку, необходим новый дайджест. Новый дайджест можно пересчитать для всей строки „AB“, но быстрее будет взять вычисленный дайджест для „A“ и внести изменения на основании добавленной части „B“. Это называется многошаговым процессом или «инкрементным» хеш-суммированием, которое поддерживает Tarantool поддерживает для всех криптографических функций.

```
crypto = require('crypto')

-- вывести дайджест 'AB' по aes-192 пошагово, затем с инкрементом
key = 'key/key/key/key/key/key/'
iv =  'iviviviviviviviv'
print(crypto.cipher.aes192.cbc.encrypt('AB', key, iv))
c = crypto.cipher.aes192.cbc.encrypt.new(key)
c:init(nil, iv)
c:update('A')
c:update('B')
print(c:result())
c:free()

-- вывести дайджест 'AB' по sha-256 пошагово, затем с инкрементом
print(crypto.digest.sha256('AB'))
c = crypto.digest.sha256.new()
c:init()
c:update('A')
c:update('B')
print(c:result())
c:free()
```

### Получение одинаковых результатов из модулей digest и crypto

Следующие функции равноценны. Например, функция digest и функция crypto приведут к одному результату.

```
crypto.cipher.aes256.cbc.encrypt('x',b32,b16)==digest.aes256cbc.encrypt('x',b32,b16)
crypto.digest.md4('string') == digest.md4('string')
crypto.digest.md5('string') == digest.md5('string')
crypto.digest.sha1('string') == digest.sha1('string')
crypto.digest.sha224('string') == digest.sha224('string')
crypto.digest.sha256('string') == digest.sha256('string')
crypto.digest.sha384('string') == digest.sha384('string')
crypto.digest.sha512('string') == digest.sha512('string')
```

### 9.5.9 Модуль csv

#### Общие сведения

Модуль `csv` обрабатывает записи, форматированные в соответствии с правилами CSV (значения, разделенные запятыми).

По умолчанию, используются следующие правила форматирования:

- Lua escape sequences such as \n or \10 are legal within strings but not within files,
- Запятые обозначают конец поля,
- Символы перевода строки или перевода строки плюс возврата каретки означают конец записи,
- Начальные и конечные пробелы игнорируются,
- Кавычками могут обрамляться поля или компоненты полей,
- При обрамлении кавычками запятые, символы перевода строки и пробелы считаются обычными символами, а двойные кавычки «» считаются одинарными.

Параметры, которые можно передать в функции модуля `csv`:

- `delimiter = строка` (по умолчанию: запятая) – однобайтовый символ для обозначения конца поля
- `quote_char = строка` (по умолчанию: кавычка) – однобайтовый символ для обозначения закрытия строки
- `chunk_size = число` (по умолчанию: 4096) – число символов для одновременного чтения (обычно для эффективности файлового ввода-вывода)
- `skip_head_lines = число` (по умолчанию: 0) – число строк, которые пропускаются в начале (обычно для заголовка)

#### Указатель

Ниже приведен перечень всех функций модуля `csv`.

| Имя | Назначение |
|---|---|
| *csv.load()* | Загрузка CSV-файла |
| *csv.dump()* | Преобразование входного значения в строку формата CSV |
| *csv.iterate()* | Итерация по записям в формате CSV |

`csv.load`(*readable*[, *{options}*])

Получение входного значения в формате CSV из `readable` и возврат таблицы в качестве выходного значения. Обычно `readable` представляет собой либо строку, либо открытый для чтения файл. Как правило, параметры *options* не указываются.

**Параметры**

- `readable` (`object`) – строка или любой объект с методом read(), форматированный по правилам CSV
- `options` (*table*) – см. *выше*

**возвращает** загруженное значение

**тип возвращаемого значения** таблица

**Пример:**

В читаемой строке 3 поля, поле №2 содержит запятую и пробел, поэтому следует использовать кавычки:

```
tarantool> csv = require('csv')
---
...
tarantool> csv.load('a,"b,c ",d')
---
- - - a
    - 'b,c '
    - d
...
```

В читаемой строке 2-байтный символ = Палочка в кириллице: (Отобразит палочку только в том случае, если кодировка = UTF-8.)

```
tarantool> csv.load('a\\211\\128b')
---
- - - a\211\128b
...
```

Точка с запятой вместо запятой в виде символа разделителя:

```
tarantool> csv.load('a,b;c,d', {delimiter = ';'})
---
- - - a,b
    - c,d
...
```

Читаемый файл `./file.csv` содержит две записи в формате CSV. Объяснение блока `fio` дается в разделе *fio*. Исходный CSV-файл и пример соответственно:

```
tarantool> -- входное значение в файле file.csv:
tarantool> -- a,"b,c ",d
tarantool> -- a\\211\\128b
tarantool> fio = require('fio')
---
...
tarantool> f = fio.open('./file.csv', {'O_RDONLY'})
---
...
tarantool> csv.load(f, {chunk_size = 4096})
---
- - - a
    - 'b,c '
    - d
  - - a\\211\\128b
...
tarantool> f:close()
---
- true
...
```

csv.dump(*csv-table*[, *options*, *writable*])

Получение входного значения из таблицы `csv-table` и возврат строки в формате CSV в качестве выходного значения. Или получение входного значения из таблицы `csv-table` и размещение выходного значения в `writable`. Обычно параметры *`options`* не указываются. Как правило, если указан `writable`, то это открытый для чтения файл. *csv.dump()* – это операция, обратная *csv.load()*.

**Параметры**

- `csv-table` (*`table`*) – таблица, которую можно форматировать в соответствии с правилами CSV
- `options` (*`table`*) – необязательно. См. *выше*
- `writable` (`object`) – любой объект с методом `write()`

**возвращает** записанное значение

**тип возвращаемого значения** строка, которая записывается в объект `writable`, если указан

**Пример:**

В таблице формата CSV 3 поля, поле №2 содержит «,» поэтому результат включает в себя кавычки

```
tarantool> csv = require('csv')
---
...
tarantool> csv.dump({'a','b,c ','d'})
---
- 'a,"b,c ",d

'
...
```

Круговое преобразование: из строки в таблицу и обратно в строку

```
tarantool> csv_table = csv.load('a,b,c')
---
...
tarantool> csv.dump(csv_table)
---
- 'a,b,c

'
...
```

csv.iterate(*input*, *{options}*)

Создание Lua-функции с итератором для прохода по записям в формате CSV по одному полю за раз. Настоятельно рекомендуется использовать итератор для большого объема данных (10 мегабайт и более).

**Параметры**

- `csv-table` (*`table`*) – таблица, которую можно форматировать в соответствии с правилами CSV
- `options` (*`table`*) – см. *выше*

**возвращает** Lua-функция с итератором

**тип возвращаемого значения** функция с итератором

**Пример:**

*csv.iterate()* is the low level of *csv.load()* and *csv.dump()*. To illustrate that, here is a function which is the same as the *csv.load()* function, as seen in the Tarantool source code.

```
tarantool> load = function(readable, opts)
         >   opts = opts or {}
         >   local result = {}
         >   for i, tup in csv.iterate(readable, opts) do
         >     result[i] = tup
         >   end
         >   return result
         > end
---
...
tarantool> load('a,b,c')
---
- - - a
    - b
    - c
...
```

### 9.5.10 Module datetime

Since *2.10.0*.

The `datetime` module provides support for the *datetime and interval data types*. It allows creating the date and time values either via the object interface or via parsing string values conforming to the ISO-8601 standard.

Below is a list of the `datetime` module functions and methods.

| Name | Use |
|---|---|
| *datetime.new()* | Create a `datetime` object from a table of time units. |
| *format()* | Convert the standard presentation of a `datetime` object into a formatted string. |
| *datetime_object:totable()* | Convert the information from a `datetime` object into the table format. |
| *set()* | Update the field values in the existing `datetime` object. |
| *parse()* | Convert an input string with the date and time information into a `datetime` object. |
| *add()* | Modify an existing datetime object by adding values of the input arguments. |
| *sub()* | Modify an existing datetime object by subtracting values of the input arguments. |
| *datetime.interval.new()* | Create an `interval` object from a table of time units. |
| *interval_object:totable()* | Convert the information from an `interval` object into the table format. |
| *Interval arithmetic* | Arithmetic operations with datetime and interval objects. |

### datetime.new()

`datetime.new(`[*{ units }*]`)`
Since *2.10.0*.

Create an object of the *datetime type* from a table of time units. See *description of units* and *examples* below.

**Параметры**

- `units` (*table*) – Table of *time units*. If an empty table or no arguments are passed, the `datetime` object with the default values corresponding to Unix Epoch is created: `1970-01-01T00:00:00Z`.

**Return** *datetime object*

**Rtype** cdata

**Possible input time units for ``datetime.new()``**

| Name | Description | Type | Default |
|---|---|---|---|
| nsec (usec, msec) | Fractional part of the last second. You can specify either nanoseconds (`nsec`), or microseconds (`usec`), or milliseconds (`msec`). Specifying two of these units simultaneously or all three ones lead to an error. | number | 0 |
| sec | Seconds. Value range: 0 - 60. | number | 0 |
| min | Minutes. Value range: 0 - 59. | number | 0 |
| hour | Hours. Value range: 0 - 23. | number | 0 |
| day | Day number. Value range: 1 - 31. The special value `-1` generates the last day of a particular month (see *example below*). | number | 1 |
| month | Month number. Value range: 1 - 12. | number | 1 |
| year | Year. | number | 1970 |
| timestamp | Timestamp, in seconds. Similar to the Unix timestamp, but can have a fractional part which is converted in nanoseconds in the resulting `datetime` object. If the fractional part for the last second is set via the `nsec`, `usec`, or `msec` units, the timestamp value should be integer otherwise an error occurs. Timestamp is not allowed if you already set time and/or date via specific units, namely, `sec`, `min`, `hour`, `day`, `month`, and `year`. | number | 0 |
| tzoffset | Time zone offset from UTC, in minutes. If both `tzoffset` and `tz` are specified, `tz` has the preference and the `tzoffset` value is ignored. | number | 0 |
| tz | Time zone name according to the tz database. | string | • |

**Examples:**

```
tarantool> datetime.new {
         nsec = 123456789,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            sec = 20,
            min = 25,
            hour = 18,

            day = 20,
            month = 8,
            year = 2021,

            tzoffset  = 180
            }
---
- 2021-08-20T18:25:20.123456789+0300
...

tarantool> datetime.new {
            nsec = 123456789,

            sec = 20,
            min = 25,
            hour = 18,

            day = 20,
            month = 8,
            year = 2021,

            tzoffset = 60,
            tz = 'Europe/Moscow'
            }
---
- 2021-08-20T18:25:20.123456789 Europe/Moscow
...

tarantool> datetime.new {
            day = -1,
            month = 2,
            year = 2021,
            }
---
- 2021-02-28T00:00:00Z
...

tarantool> datetime.new {
            timestamp = 1656664205.123,
            tz = 'Europe/Moscow'
            }
---
- 2022-07-01T08:30:05.122999906 Europe/Moscow
...

tarantool> datetime.new {
            nsec = 123,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            timestamp = 1656664205,
            tz = 'Europe/Moscow'
            }
---
- 2022-07-01T08:30:05.000000123 Europe/Moscow
...
```

### datetime_object

`object datetime_object`

Since *2.10.0*.

`datetime_object:totable()`

Convert the information from a `datetime` object into the table format. Resulting table has the following fields:

| Field name | Description |
|---|---|
| nsec | Nanosecods |
| sec | Seconds |
| min | Minutes |
| hour | Hours |
| day | Day number |
| month | Month number |
| year | Year |
| wday | Days since the beginning of the week |
| yday | Days since the beginning of the year |
| isdst | Is the DST (Daylight saving time) applicable for the date. Boolean. |
| tzoffset | Time zone offset from UTC |

**Return** table with the date and time parameters

**Rtype** *table*

**Example:**

```
tarantool> dt = datetime.new {
            sec = 20,
            min = 25,
            hour = 18,

            day = 20,
            month = 8,
            year = 2021,
            }
---
...

tarantool> dt:totable()
---
- sec: 20
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  min: 25
  yday: 232
  day: 20
  nsec: 0
  isdst: false
  wday: 6
  tzoffset: 0
  month: 8
  year: 2021
  hour: 18
...
```

`datetime_object:format([`*'input_string'*`])`

Convert the standard `datetime` object presentation into a formatted string. The conversion specifications are the same as in the strftime library. Additional specification for nanoseconds is `%f` which also allows a modifier to control the output precision of fractional part: `%5f` (see the example below). If no arguments are set for the method, the default conversions are used: `'%FT%T.%f%z'` (see the example below).

**Параметры**

- `input_string` (*string*) – string consisting of zero or more conversion specifications and ordinary characters

**Return** string with the formatted date and time information

**Rtype** *string*

**Example:**

```
tarantool> dt = datetime.new {
           nsec = 123456789,

           sec = 20,
           min = 25,
           hour = 18,

           day = 20,
           month = 8,
           year = 2021,

           tzoffset  = 180
           }
---
...

tarantool> dt:format('%d.%m.%y %H:%M:%S.%5f')
---
- 20.08.21 18:25:20.12345
...

tarantool> dt:format()
---
- 2021-08-20T18:25:20.123456789+0300
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...

tarantool> dt:format('%FT%T.%f%z')
---
- 2021-08-20T18:25:20.123456789+0300
...
```

datetime_object:set([*{ units }*])

Update the field values in the existing `datetime` object.

**Параметры**

- `units` (*table*) – Table of time units. The *time units* are the same as for the `datetime.new()` function.

**Return** updated datetime_object

**Rtype** cdata

**Example:**

```
tarantool> dt = datetime.new {
           nsec = 123456789,

           sec = 20,
           min = 25,
           hour = 18,

           day = 20,
           month = 8,
           year = 2021,

           tzoffset  = 180
           }

tarantool> dt:set {msec = 567}
---
- 2021-08-20T18:25:20.567+0300
...

tarantool> dt:set {tzoffset = 60}
---
- 2021-08-20T18:25:20.567+0100
...
```

datetime_object:parse(*'input_string'*[, *{format, tzoffset}*])

Convert an input string with the date and time information into a `datetime` object. The input string should be formatted according to one of the following standards:

- ISO 8601
- RFC 3339
- extended strftime – see description of the *format()* for details.

**Параметры**

- `input_string` (*string*) – string with the date and time information.
- `format` (*string*) – indicator of the input_sting format. Possible values: „iso8601“, „rfc3339“, or `strptime`-like format string. If no value is set, the default formatting is used.
- `tzoffset` (`number`) – time zone offset from UTC, in minutes.

**Return** a datetime_object

**Rtype** cdata

**Example:**

```
tarantool> t = datetime.parse('1970-01-01T00:00:00Z')

tarantool> t
---
- 1970-01-01T00:00:00Z
...

tarantool> t = datetime.parse('1970-01-01T00:00:00', {format = 'iso8601',␣
↪tzoffset = 180})

tarantool> t
---
- 1970-01-01T00:00:00+0300
...

tarantool> t = datetime.parse('2017-12-27T18:45:32.999999-05:00', {format =
↪'rfc3339'})

tarantool> t
---
- 2017-12-27T18:45:32.999999-0500
...

tarantool> T = datetime.parse('Thu Jan  1 03:00:00 1970', {format = '%c'})

tarantool> T
---
- 1970-01-01T03:00:00Z
...

tarantool> T = datetime.parse('12/31/2020', {format = '%m/%d/%y'})

tarantool> T
---
- 2020-12-31T00:00:00Z
...

tarantool> T = datetime.parse('1970-01-01T03:00:00.125000000+0300', {format = '
↪%FT%T.%f%z'})

tarantool> T
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
- 1970-01-01T03:00:00.125+0300
...
```

datetime_object:add(*input*[, *{ adjust }*])

Modify an existing datetime object by adding values of the input argument.

**Параметры**

- `input` (*table*) – an *interval object* or an equivalent table (see **Example #1**)
- `adjust` (*string*) – defines how to round days in a month after an arithmetic operation. Possible values: `none`, `last`, `excess` (see **Example #2**). Defaults to `none`.

**Return** datetime_object

**Rtype** cdata

**Example #1:**

```
tarantool> dt = datetime.new {
            day = 26,
            month = 8,
            year = 2021,
            tzoffset  = 180
            }
---
...

tarantool> iv = datetime.interval.new {day = 7}
---
...

tarantool> dt, iv
---
- 2021-08-26T00:00:00+0300
- +7 days
...

tarantool> dt:add(iv)
---
- 2021-09-02T00:00:00+0300
...

tarantool> dt:add{ day = 7 }
---
- 2021-09-09T00:00:00+0300
...
```

**Example #2:**

```
tarantool> dt = datetime.new {
            day = 29,
            month = 2,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            year = 2020
            }
---
...

tarantool> dt:add{month = 1, adjust = 'none'}
---
- 2020-03-29T00:00:00Z
...

tarantool> dt = datetime.new {
            day = 29,
            month = 2,
            year = 2020
            }
---
...

tarantool> dt:add{month = 1, adjust = 'last'}
---
- 2020-03-31T00:00:00Z
...

tarantool> dt = datetime.new {
            day = 31,
            month = 1,
            year = 2020
            }
---
...

tarantool> dt:add{month = 1, adjust = 'excess'}
---
- 2020-03-02T00:00:00Z
...
```

`datetime_object:sub(`*{ input*[, *adjust* ] *}*`)`

Modify an existing datetime object by subtracting values of the input argument.

**Параметры**

- `input` (*table*) – an *interval object* or an equivalent table (see **Example**)
- `adjust` (*string*) – defines how to round days in a month after an arithmetic operation. Possible values: `none`, `last`, `excess`. Defaults to `none`. The logic is similar to the one of the `:add()` method – see *Example #2*.

**Return** datetime_object

**Rtype** cdata

**Example:**

```
tarantool> dt = datetime.new {
            day = 26,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            month = 8,
            year = 2021,
            tzoffset  = 180
            }
---
...

tarantool> iv = datetime.interval.new {day = 5}
---
...

tarantool> dt, iv
---
- 2021-08-26T00:00:00+0300
- +5 days
...

tarantool> dt:sub(iv)
---
- 2021-08-21T00:00:00+0300
...

tarantool> dt:sub{ day = 1 }
---
- 2021-08-20T00:00:00+0300
...
```

### datetime.interval.new()

`datetime.interval.new(`$\left[\{\ input\ \}\right]$`)`

Since *2.10.0*.

Create an object of the *interval type* from a table of time units. See *description of units* and *examples* below.

**Параметры**

- `input` (*table*) – Table with *time units and parameters*. For all possible time units, the values are not restricted. If an empty table or no arguments are passed, the `interval` object with the default value `0 seconds` is created.

**Return** interval_object

**Rtype** cdata

**Possible input time units and parameters for ``datetime.interval.new()**

| Name | Description | Type | Default |
|---|---|---|---|
| nsec (usec, msec) | Fractional part of the last second. You can specify either nanoseconds (`nsec`), or microseconds (`usec`), or milliseconds (`msec`). Specifying two of these units simultaneously or all three ones lead to an error. | number | 0 |
| sec | Seconds | number | 0 |
| min | Minutes | number | 0 |
| hour | Hours | number | 0 |
| day | Day number | number | 0 |
| week | Week number | number | 0 |
| month | Month number | number | 0 |
| year | Year | number | 0 |
| adjust | Defines how to round days in a month after an arithmetic operation. | string | „none“ |

**Examples:**

```
tarantool> datetime.interval.new()

---
- 0 seconds
...

tarantool> datetime.interval.new {
            month = 6,
            year = 1
            }
---
- +1 years, 6 months
...

tarantool> datetime.interval.new {
            day = -1
            }
---
- -1 days
...
```

### interval_object

`object interval_object`

Since *2.10.0*.

`interval_object:totable()`

Convert the information from an `interval` object into the table format. Resulting table has the following fields:

| Field name | Description |
|---|---|
| nsec | Nanosecods |
| sec | Seconds |
| min | Minutes |
| hour | Hours |
| day | Day number |
| month | Month number |
| year | Year |
| week | Week number |
| adjust | Defines how to round days in a month after an arithmetic operation. |

**Return** table with the date and time parameters

**Rtype** *table*

**Example:**

```
tarantool> iv = datetime.interval.new{month = 1, adjust = 'last'}
---
...

tarantool> iv:totable()
---
- adjust: last
  sec: 0
  nsec: 0
  day: 0
  week: 0
  hour: 0
  month: 1
  year: 0
  min: 0
...
```

### Interval arithmetic

Since *2.10.0*.

The *datetime module* enables creating objects of two types: `datetime` and `interval`.

If you need to shift the `datetime` object values, you can use either the modifier methods, that is, the *:add* or *:sub* methods, or apply interval arithmetic using overloaded `+ (__add)` or `- (__sub)` methods.

`:add`/`:sub` modify the current object, but `+`/`-` create copy of the object as the operation result.

In the interval operation, each of the interval subcomponents are sequentially calculated starting from the largest (`year`) to the smallest (`nsec`):

- `year` – years
- `month` – months
- `week` – weeks
- `day` – days

- `hour` – hours
- `min` – minutes
- `sec` – seconds
- `nsec` – nanoseconds.

If results of the operation exceed the allowed range for any of the components, an exception is raised.

The `datetime` and `interval` objects can participate in arithmetic operations:

- The sum of two intervals is an interval object, which fields are sum of each particular component of operands.
- The result of subtraction of two intervals is similar: it's an interval object where each subcomponent is the result of subtraction of particular fields in the original operands.
- If you add datetime and interval objects, the result is a datetime object. Addition is being performed in a determined order from the largest component (`year`) to the smallest (`nsec`).
- Subtraction of two datetime objects produce an interval object. Difference of two time values is performed not as difference of the epoch seconds, but as difference of all the subcomponents, that is, years, months, days, hours, minutes, and seconds.
- An untyped table object can be used in each context where the typed datetime or interval objects are used if left operand is typed object with overloaded operation of `+` or `-`.

The matrix of the `addition` operands eligibility and their result types:

| | datetime | interval | table |
|---|---|---|---|
| **datetime** | | datetime | datetime |
| **interval** | datetime | interval | interval |
| **table** | | | |

The matrix of the `subtraction` operands eligibility and their result types:

| | datetime | interval | table |
|---|---|---|---|
| **datetime** | interval | datetime | datetime |
| **interval** | | interval | interval |
| **table** | | | |

### 9.5.11 Модуль decimal

The `decimal` module has functions for working with exact numbers. This is important when numbers are large or even the slightest inaccuracy is unacceptable. For example Lua calculates `0.16666666666667 * 6` with floating-point so the result is 1. But with the decimal module (using `decimal.new` to convert the number to decimal type) `decimal.new('0.16666666666667') * 6` is 1.00000000000002.

To construct a decimal number, bring in the module with `require('decimal')` and then use `decimal.new(n)` or any function in the decimal module:

- *abs(n)*
- *exp(n)*
- *is_decimal(n)*
- *ln(n)*

- *log10(n)*
- *new(n)*
- *precision(n)*
- *rescale(decimal-number, new-scale)*
- *scale(n)*
- *sqrt(n)*
- *trim(decimal-number)*,

where n can be a string or a non-decimal number or a decimal number. If it is a string or a non-decimal number, Tarantool converts it to a decimal number before working with it. It is best to construct from strings, and to convert back to strings after calculations, because Lua numbers have only 15 digits of precision. Decimal numbers have 38 digits of precision, that is, the total number of digits before and after the decimal point can be 38. Tarantool supports the usual arithmetic and comparison operators + - * / % ^ < > <= >= ~= ==. If an operation has both decimal and non-decimal operands, then the non-decimal operand is converted to decimal before the operation happens.

Use `tostring(decimal-number)` to convert back to a string.

A decimal operation will fail if overflow happens (when a number is greater than 10^38 - 1 or less than -10^38 - 1). A decimal operation will fail if arithmetic is impossible (such as division by zero or square root of minus 1). A decimal operation will not fail if rounding of post-decimal digits is necessary to get 38-digit precision.

`decimal.abs(`*string-or-number-or-decimal-number*`)`
: Returns absolute value of a decimal number. For example if `a` is `-1` then `decimal.abs(a)` returns `1`.

`decimal.exp(`*string-or-number-or-decimal-number*`)`
: Returns *e* raised to the power of a decimal number. For example if `a` is `1` then `decimal.exp(a)` returns `2.71828182845904523536028747135266249 78`. Compare `math.exp(1)` from the Lua math library, which returns `2.718281828459`.

`decimal.is_decimal(`*string-or-number-or-decimal-number*`)`
: Returns `true` if the specified value is a decimal, and `false` otherwise. For example if `a` is `123` then `decimal.is_decimal(a)` returns `false`. if `a` is `decimal.new(123)` then `decimal.is_decimal(a)` returns `true`.

`decimal.ln(`*string-or-number-or-decimal-number*`)`
: Returns natural logarithm of a decimal number. For example if `a` is `1` then `decimal.ln(a)` returns `0`.

`decimal.log10(`*string-or-number-or-decimal-number*`)`
: Returns base-10 logarithm of a decimal number. For example if `a` is `100` then `decimal.log10(a)` returns `2`.

`decimal.new(`*string-or-number-or-decimal-number*`)`
: Returns the value of the input as a decimal number. For example if `a` is `1E-1` then `decimal.new(a)` returns `0.1`.

`decimal.precision(`*string-or-number-or-decimal-number*`)`
: Returns the number of digits in a decimal number. For example if `a` is `123.4560` then `decimal.precision(a)` returns `7`.

`decimal.rescale(`*decimal-number*, *new-scale*`)`
: Returns the number after possible rounding or padding. If the number of post-decimal digits is greater than new-scale, then rounding occurs. The rounding rule is: round half away from zero. If the number of post-decimal digits is less than new-scale, then padding of zeros occurs. For example if `a` is `-123.4550` then `decimal.rescale(a, 2)` returns `-123.46`, and `decimal.rescale(a, 5)` returns `-123.45500`.

`decimal.scale(`*string-or-number-or-decimal-number*`)`
Returns the number of post-decimal digits in a decimal number. For example if `a` is `123.4560` then `decimal.scale(a)` returns 4.

`decimal.sqrt(`*string-or-number-or-decimal-number*`)`
Returns the square root of a decimal number. For example if `a` is `2` then `decimal.sqrt(a)` returns `1.414213562373095048801688724209698078`6.

`decimal.trim(`*decimal-number*`)`
Returns a decimal number after possible removing of trailing post-decimal zeros. For example if `a` is `2.20200` then `decimal.trim(a)` returns `2.202`.

### 9.5.12 Модуль digest

#### Общие сведения

A «digest» is a value which is returned by a function (usually a Cryptographic hash function), applied against a string. Tarantool's `digest` module supports several types of cryptographic hash functions ( AES, MD4, MD5, SHA-1, SHA-2, PBKDF2) as well as a checksum function (CRC32), two functions for base64, and two non-cryptographic hash functions (guava, murmur). Some of the digest functionality is also present in the *crypto*.

#### Указатель

Ниже приведен перечень всех функций модуля `digest`.

| Имя | Назначение |
|---|---|
| *digest.aes256cbc.encrypt()* | Шифрование строки с использованием AES |
| *digest.aes256cbc.decrypt()* | Расшифрование строки с использованием AES |
| *digest.md4()* | Получение дайджеста с помощью MD4 |
| *digest.md4_hex()* | Получение шестнадцатеричного дайджеста с помощью MD4 |
| *digest.md5()* | Получение дайджеста с помощью MD5 |
| *digest.md5_hex()* | Получение шестнадцатеричного дайджеста с помощью MD5 |
| *digest.pbkdf2()* | Получение дайджеста с помощью PBKDF2 |
| *digest.sha1()* | Получение дайджеста с помощью SHA-1 |
| *digest.sha1_hex()* | Получение шестнадцатеричного дайджеста с помощью SHA-1 |
| *digest.sha224()* | Получение 224-битного дайджеста с помощью SHA-2 |
| *digest.sha224_hex()* | Получение 56-байтного шестнадцатеричного дайджеста с помощью SHA-2 |
| *digest.sha256()* | Получение 256-битного дайджеста с помощью SHA-2 |
| *digest.sha256_hex()* | Получение 64-байтного шестнадцатеричного дайджеста с помощью SHA-2 |
| *digest.sha384()* | Получение 384-битного дайджеста с помощью SHA-2 |
| *digest.sha384_hex()* | Получение 96-байтного шестнадцатеричного дайджеста с помощью SHA-2 |
| *digest.sha512()* | Получение 512-битного дайджеста с помощью SHA-2 |
| *digest.sha512_hex()* | Получение 128-байтного шестнадцатеричного дайджеста с помощью SHA-2 |
| *digest.base64_encode()* | Кодирование строки по стандарту Base64 |
| *digest.base64_decode()* | Декодирование строки по стандарту Base64 |
| *digest.urandom()* | Получение массива случайных байтов |
| *digest.crc32()* | Получение 32-битной контрольной суммы с помощью CRC32 |
| *digest.crc32.new()* | Запуск инкрементного вычисления CRC32 |
| *digest.guava()* | Получение числа с помощью консистентного хеширования |
| *digest.murmur()* | Получение дайджеста с помощью MurmurHash |
| *digest.murmur.new()* | Запуск инкрементного вычисления с помощью MurmurHash |

`digest.aes256cbc.encrypt(`*string*, *key*, *iv*`)`
`digest.aes256cbc.decrypt(`*string*, *key*, *iv*`)`

Возврат 256-битной двоичной строки = дайджест, полученный с помощью AES.

`digest.md4(`*string*`)`

Возврат 128-битной двоичной строки = дайджест, полученный с помощью MD4.

`digest.md4_hex(`*string*`)`

Возврат 32-байтной строки = шестнадцатеричное значение дайджеста, вычисленного с помощью MD4.

`digest.md5(`*string*`)`

Возврат 128-битной двоичной строки = дайджест, полученный с помощью MD5.

`digest.md5_hex(`*string*`)`

Возврат 32-байтной строки = шестнадцатеричное значение дайджеста, вычисленного с помощью MD5.

`digest.pbkdf2(`*string*, *salt*[, *iterations*[, *digest-length*]])

Возврат двоичной строки = дайджест, полученный с помощью PBKDF2. Для эффективности шифрования значение параметра количества итераций `iterations` должно быть как минимум несколько тысяч. Значение параметра `digest-length` определяет длину полученной двоичной строки.

**Примечание:** `digest.pbkdf2()` yields and should not be used in a transaction (between `box.begin()` and `box.commit()`/`box.rollback()`). PBKDF2 is a time-consuming hash algorithm. It runs in a separate coio thread. While calculations are performed, the fiber that calls `digest.pbkdf2()` yields and another fiber continues working in the tx thread.

`digest.sha1(`*string*`)`
Возврат 160-битной двоичной строки = дайджест, полученный с помощью SHA-1.

`digest.sha1_hex(`*string*`)`
Возврат 40-байтной строки = шестнадцатеричное значение дайджеста, вычисленного с помощью SHA-1.

`digest.sha224(`*string*`)`
Возврат 224-битной двоичной строки = дайджест, полученный с помощью SHA-2.

`digest.sha224_hex(`*string*`)`
Возврат 56-байтной строки = шестнадцатеричное значение дайджеста, вычисленного с помощью SHA-224.

`digest.sha256(`*string*`)`
Возврат 256-битной двоичной строки = дайджест, полученный с помощью SHA-2.

`digest.sha256_hex(`*string*`)`
Возврат 64-байтной строки = шестнадцатеричное значение дайджеста, вычисленного с помощью SHA-256.

`digest.sha384(`*string*`)`
Возврат 384-битной двоичной строки = дайджест, полученный с помощью SHA-2.

`digest.sha384_hex(`*string*`)`
Возврат 96-байтной строки = шестнадцатеричное значение дайджеста, вычисленного с помощью SHA-384.

`digest.sha512(`*string*`)`
Возврат 512-битной двоичной строки = дайджест, полученный с помощью SHA-2.

`digest.sha512_hex(`*string*`)`
Возврат 128-байтной строки = шестнадцатеричное значение дайджеста, вычисленного с помощью SHA-512.

`digest.base64_encode()`
Возврат кодированного по base64 значения обычной строки.

Возможные опции:

- `nopad` – результат не должен включать в себя „=“ для заполнения символами в конце,
- `nowrap` – результат не должен включать в себя символ переноса строки для разделения строк после 72 символов,
- `urlsafe` – результат не должен включать в себя „=“ или символ переноса строки и может содержать „-„ или „_“ взамен „+“ или „/“ в качестве 62 и 63 символа в схеме.

Значения параметров могут быть `true` (правда) или `false` (ложь), по умолчанию используется `false`.

Пример:

```
digest.base64_encode(string_variable,{nopad=true})
```

digest.base64_decode(*string*)
: Возврат обычной строки из кодированного по base64 значения.

digest.urandom(*integer*)
: Возврат массива случайных байтов с длиной = целому числу.

digest.crc32(*string*)
: Возврат 32-битной контрольной суммы с помощью CRC32.

  The `crc32` and `crc32_update` functions use the Cyclic Redundancy Check polynomial value: `0x1EDC6F41` / `4812730177`. (Other settings are: input = reflected, output = reflected, initial value = 0xFFFFFFFF, final xor value = 0x0.) If it is necessary to be compatible with other checksum functions in other programming languages, ensure that the other functions use the same polynomial value.

  Например, в Python установите пакет `crcmod` и введите команду:

```
>>> import crcmod
>>> fun = crcmod.mkCrcFun('4812730177')
>>> fun('string')
3304160206L
```

  В Perl установите модуль `Digest::CRC` и выполните следующий код:

```
use Digest::CRC;
$d = Digest::CRC->new(width => 32, poly => 0x1EDC6F41, init => 0xFFFFFFFF, refin =>␣
↪1, refout => 1);
$d->add('string');
print $d->digest;
```

  (ожидается выходное значение: 3304160206).

digest.crc32.new()
: Запуск инкрементного вычисления CRC32. См. примечания по *инкрементным методам*.

digest.guava(*state*, *bucket*)
: Возврат числа с помощью консистентного хеширования.

  The guava function uses the Consistent Hashing algorithm of the Google guava library. The first parameter should be a hash code; the second parameter should be the number of buckets; the returned value will be an integer between 0 and the number of buckets. For example,

```
tarantool> digest.guava(10863919174838991, 11)
---
- 8
...
```

digest.murmur(*string*)
: Возврат 32-битной двоичной строки = дайджест, полученный с помощью MurmurHash.

digest.murmur.new(*opts*)
: Запуск инкрементного вычисления с помощью MurmurHash. См. примечания по *инкрементным методам*. Например:

```
murmur.new({seed=0})
```

### Инкрементальные методы в модуле digest

Предположим, что вычислен дайджест для строки „A“, затем часть „B“ добавляется в строку, необходим новый дайджест. Новый дайджест можно пересчитать для всей строки „AB“, но быстрее будет взять вычисленный дайджест для „A“ и внести изменения на основании добавленной части „B“. Это называется многошаговым процессом или «инкрементным» хеш-суммированием, которое поддерживает Tarantool поддерживает для crc32 и murmur...

```
digest = require('digest')

-- вывести дайджест 'AB' по crc32 пошагово, затем с инкрементом
print(digest.crc32('AB'))
c = digest.crc32.new()
c:update('A')
c:update('B')
print(c:result())

-- вывести дайджест 'AB' по murmur hash пошагово, затем с инкрементом
print(digest.murmur('AB'))
m = digest.murmur.new()
m:update('A')
m:update('B')
print(m:result())
```

### Пример

In the following example, the user creates two functions, `password_insert()` which inserts a SHA-1 digest of the word «**^S^e^c^ret Wordpass**» into a tuple set, and `password_check()` which requires input of a password.

```
tarantool> digest = require('digest')
---
...
tarantool> function password_insert()
         >   box.space.tester:insert{1234, digest.sha1('^S^e^c^ret Wordpass')}
         >   return 'OK'
         > end
---
...
tarantool> function password_check(password)
         >   local t = box.space.tester:select{12345}
         >   if digest.sha1(password) == t[2] then
         >     return 'Password is valid'
         >   else
         >     return 'Password is not valid'
         >   end
         > end
---
...
tarantool> password_insert()
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- 'OK'
...
```

Если затем пользователь вызовет функцию `password_check()` и вводит неверный пароль, результатом будет ошибка.

```
tarantool> password_check('Secret Password')
---
- 'Password is not valid'
...
```

### 9.5.13 Модуль errno

#### Общие сведения

Модуль `errno`, как правило, используется внутри функции или в рамках Lua-программы совместно с модулем, функции которого могут возвращать ошибки ОС, например *fio*.

#### Указатель

Ниже приведен перечень всех функций модуля `errno`.

| Имя | Назначение |
|---|---|
| *errno()* | Получение номера ошибки для последней функции, связанной с ОС |
| *errno.strerror()* | Получение сообщения об ошибке для соответствующего номера ошибки |

`errno()`

Возврат номера ошибки для последней функции, связанной с операционной системой, или 0. Чтобы вызвать функцию, просто введите команду `errno()` без названия модуля.

**тип возвращаемого значения** целое число

`errno.strerror(`[*code*]`)`

Возврат строки в ответ на номер ошибки. Строка будет содержать текст традиционного сообщения об ошибке для текущей операционной системы. Если не указан код `code`, то будет выведено сообщение об ошибке для последней функции, связанной с операционной системой, или 0.

**Параметры**

- `code` (`integer`) – номер ошибки в операционной системе

**тип возвращаемого значения** строка

**Пример:**

Данная функция отображает результат вызова *fio.open()*, который вызывает ошибку 2 (`errno.ENOENT`). В результат включен номер ошибки, связанная с ним строка сообщения об ошибке и название ошибки.

```
tarantool> function f()
         >   local fio = require('fio')
         >   local errno = require('errno')
         >   fio.open('no_such_file')
         >   print('errno() = ' .. errno())
```

(continues on next page)

(продолжение с предыдущей страницы)

```
         >   print('errno.strerror() = ' .. errno.strerror())
         >   local t = getmetatable(errno).__index
         >   for k, v in pairs(t) do
         >     if v == errno() then
         >       print('errno() constant = ' .. k)
         >     end
         >   end
         > end
---
...

tarantool> f()
errno() = 2
errno.strerror() = No such file or directory
errno() constant = ENOENT
---
...
```

Чтобы увидеть все возможные названия ошибок, которые хранятся в метатаблице `errno`, введите команду `getmetatable(errno)` (выводятся сокращенно):

```
tarantool> getmetatable(errno)
---
- __newindex: 'function: 0x41666a38'
  __call: 'function: 0x41666890'
  __index:
  ENOLINK: 67
  EMSGSIZE: 90
  EOVERFLOW: 75
  ENOTCONN: 107
  EFAULT: 14
  EOPNOTSUPP: 95
  EEXIST: 17
  ENOSR: 63
  ENOTSOCK: 88
  EDESTADDRREQ: 89
  <...>
...
```

### 9.5.14 Module fiber

#### Общие сведения

С помощью модуля `fiber` можно:

- создавать, запускать и управлять *файберами*;
- отправлять и получать сообщения для различных процессов (например, разные соединения, сессии или файберы) по *каналам*;
- использовать *механизм синхронизации* для файберов, аналогично работе «условных переменных» и функций операционных систем, таких как `pthread_cond_wait()` плюс `pthread_cond_signal()`.

### Указатель

Ниже приведен перечень всех функций и элементов модуля `fiber`.

| Имя | Назначение |
|---|---|
| **Fibers** | |
| *fiber.create()* | Создание и запуск файбера |
| *fiber.new()* | Создание файбера без запуска |
| *fiber.self()* | Получение объекта файбера |
| *fiber.find()* | Получение объекта файбера по ID |
| *fiber.sleep()* | Перевод файбера в режим ожидания |
| *fiber.yield()* | Передача управления |
| *fiber.status()* | Получение статуса активного файбера |
| *fiber.info()* | Получение информации о всех файберах |
| *fiber.top()* | Возврат таблицы с активными файберами и отображение статист |
| *fiber.kill()* | Отмена файбера |
| *fiber.testcancel()* | Проверка отмены действующего файбера |
| *fiber.set_max_slice()* | Set the default maximum slice for all fibers |
| *fiber.set_slice()* | Set a slice for the current fiber execution |
| *fiber.extend_slice()* | Extend a slice for the current fiber execution |
| *fiber.check_slice()* | Check whether a slice for the current fiber is over |
| *fiber.time()* | Получение системного времени в секундах |
| *fiber.time64()* | Получение системного времени в микросекундах |
| *fiber.clock()* | Получение монотонного времени в секундах |
| *fiber.clock64()* | Получение монотонного времени в микросекундах |
| **Fiber object** | |
| *fiber_object:id()* | Получение ID файбера |
| *fiber_object:name()* | Получение имени файбера |
| *fiber_object:name(name)* | Назначение имени файбера |
| *fiber_object:status()* | Получение статуса файбера |
| *fiber_object:cancel()* | Отмена файбера |
| *fiber_object.set_max_slice()* | Set a fiber's maximum slice |
| *fiber_object.storage* | Локальное хранилище в пределах файбера |
| *fiber_object:set_joinable()* | Создание возможности подключения нового файбера |
| *fiber_object:join()* | Ожидание статуса „dead“ (недоступен) для файбера |
| **Channels** | |
| *fiber.channel()* | Создание канала связи |
| *channel_object:put()* | Отправка сообщения по каналу связи |
| *channel_object:close()* | Закрытие канала |
| *channel_object:get()* | Перехват сообщения из канала |
| *channel_object:is_empty()* | Проверка пустоты канала |
| *channel_object:count()* | Подсчет сообщений в канале |
| *channel_object:is_full()* | Проверка заполненности канала |
| *channel_object:has_readers()* | Проверка пустого канала на наличие читателей в состоянии ожи |
| *channel_object:has_writers()* | Проверка полного канала на наличие писателей в состоянии ожи |
| *channel_object:is_closed()* | Проверка закрытия канала |
| *Example* | A useful example about channels |
| **Condition variables** | |
| *fiber.cond()* | Создание условной переменной |
| *cond_object:wait()* | Перевод файбера в режим ожидания до пробуждения другим фа |
| *cond_object:signal()* | Пробуждение отдельного файбера |

Таблица 4 – продолжение с предыдущей страницы

| Имя | Назначение |
|---|---|
| *cond_ object:broadcast()* | Пробуждение всех файберов |
| *Example* | A useful example about condition variables |

### Файберы

A *fiber* is a set of instructions that are executed with cooperative multitasking. The `fiber` module enables you to *create a fiber* and associate it with a user-supplied function called a *fiber function*.

A fiber has the following possible states: `running`, `suspended`, `ready`, or `dead`. A program with fibers is, at any given time, running only one of its fibers. This running fiber only suspends its execution when it explicitly *yields control* to another fiber that is ready to execute.

When the fiber function ends, the fiber ends and becomes `dead`. If required, you can *cancel* a running or suspended fiber. Another useful capability is *limiting* a fiber execution time for long-running operations.

---

**Примечание:** By default, each *transaction* in Tarantool is executed in a single fiber on a single thread, sees a consistent database state, and commits all changes atomically.

---

### Create a fiber

To create a fiber, call one of the following functions:

- *fiber.create()* creates a fiber and runs it immediately. The initial fiber state is `running`.
- *fiber.new()* creates a fiber but does not start it. The initial fiber state is `ready`. You can join such fibers by calling the *fiber_ object:join()* function and get the result returned by the fiber's function.

### Передача управления

*Yield* is an action that occurs in a cooperative environment that transfers control of the thread from the current fiber to another fiber that is ready to execute. The `fiber` module provides the following functions that yield control to another fiber explicitly:

- *fiber.yield()* yields control to the scheduler.
- *fiber.sleep()* yields control to the scheduler and sleeps for the specified number of seconds.

### Отмена файбера

To cancel a fiber, use the *fiber_ object.cancel* function. You can also call *fiber.kill()* to locate a fiber by its numeric ID and cancel it.

### Limit execution time

If a fiber works too long without yielding control, you can use a fiber slice to limit its execution time. The *fiber_slice_default* `compat` option controls the default value of the maximum fiber slice.

There are two slice types: a warning and an error slice.

- When a warning slice is over, a *warning* message is logged, for example:

```
fiber has not yielded for more than 0.500 seconds
```

- When an error slice is over, the fiber is cancelled and the `FiberSliceIsExceeded` error is thrown:

```
FiberSliceIsExceeded: fiber slice is exceeded
```

  Control is passed to another fiber that is ready to execute.

The fiber slice is checked by all functions operating on spaces and indexes, such as *index_object.select()*, *space_object.replace()*, and so on. You can also use the *fiber.check_slice()* function in application code to check whether the slice for the current fiber is over.

The following functions override the the default value of the maximum fiber slice:

- *fiber.set_max_slice(slice)* sets the default maximum slice for all fibers.
- *fiber_object:set_max_slice(slice)* sets the maximum slice for a particular fiber.

The maximum slice is set when a fiber wakes up. This might be its first run or wake up after `fiber.yield()`.

You can change or increase the slice for a current fiber's execution using the following functions:

- *fiber.set_slice(slice)* sets the slice for a current fiber execution.
- *fiber.extend_slice(slice)* extends the slice for a current fiber execution.

Note that the specified values don't affect a fiber's execution after `fiber.yield()`.

### Information about fibers

To get information about all fibers or a specific fiber, use the following functions:

- *fiber.info* returns information about all fibers.
- *fiber.status()* gets the current fiber's status. To get the status of the specified fiber, call *fiber_object:status()*.
- *fiber.top()* shows all alive fibers and their CPU consumption.

### Garbage collection

Сборщик мусора собирает недоступные файберы так же, как и все Lua-объекты: сборщик мусора в Lua освобождает память выделенного для файбера пула, сбрасывает все данные файбера и возвращает файбер (который теперь называется каркасом файбера) в пул файберов. Каркас можно использовать повторно при создании другого файбера.

A fiber has all the features of a Lua coroutine and all the programming concepts that apply to Lua coroutines apply to fibers as well. However, Tarantool has made some enhancements for fibers and has used fibers internally. So, although the use of coroutines is possible and supported, the use of fibers is recommended.

### API reference

`fiber.create`(*function*[, *function-arguments*])

Создание и запуск файбера. Происходит создание файбера, который незамедлительно начинает работу.

**Параметры**

- `function` – функция, которая будет связана с файбером
- `function-arguments` – arguments to be passed to the function

**возвращает** созданный объект файбера

**тип возвращаемого значения** пользовательские данные

**Пример:**

The script below shows how to create a fiber using `fiber.create`:

```
-- app.lua --
fiber = require('fiber')

function greet(name)
    print('Hello, '..name)
end

greet_fiber = fiber.create(greet, 'John')
print('Fiber already started')
```

The following output should be displayed after *running* `app.lua`:

```
$ tarantool app.lua
Hello, John
Fiber already started
```

`fiber.new`(*function*[, *function-arguments*])

Create a fiber but do not start it. The created fiber starts after the fiber creator (that is, the job that is calling `fiber.new()`) yields. The initial fiber state is `ready`.

---

**Примечание:** Note that *fiber.status()* returns the `suspended` state for `ready` fibers because the `ready` state is not observable using the `fiber` module API.

---

You can join fibers created using `fiber.new` by calling the *fiber_object:join()* function and get the result returned by the fiber's function. To join the fiber, you need to make it joinable using *fiber_object:set_joinable()*.

**Параметры**

- `function` – функция, которая будет связана с файбером
- `function-arguments` – arguments to be passed to the function

**возвращает** созданный объект файбера

**тип возвращаемого значения** пользовательские данные

**Пример:**

The script below shows how to create a fiber using `fiber.new`:

```
-- app.lua --
fiber = require('fiber')

function greet(name)
    print('Hello, '..name)
end

greet_fiber = fiber.new(greet, 'John')
print('Fiber not started yet')
```

The following output should be displayed after *running* `app.lua`:

```
$ tarantool app.lua
Fiber not started yet
Hello, John
```

`fiber.self()`

**возвращает** объект файбера для запланированного на данный момент файбера.

**тип возвращаемого значения** пользовательские данные

**Пример:**

```
tarantool> fiber.self()
---
- status: running
  name: interactive
  id: 101
...
```

`fiber.find(`*id*`)`

**Параметры**

- `id` – числовой идентификатор файбера.

**возвращает** объект файбера для указанного файбера.

**тип возвращаемого значения** пользовательские данные

**Пример:**

```
tarantool> fiber.find(101)
---
- status: running
  name: interactive
  id: 101
...
```

`fiber.sleep(`*time*`)`

Передача управления планировщику и переход в режим ожидания на указанное количество секунд. Только текущий файбер можно перевести в режим ожидания.

**Параметры**

- `time` – количество секунд в режиме ожидания.

**Исключение** см. *Пример неудачной передачи управления*

**Пример:**

The `increment` function below contains an infinite loop that adds 1 to the `counter` global variable. Then, the current fiber goes to sleep for `period` seconds. `sleep` causes an implicit *fiber.yield()*.

```
-- app.lua --
fiber = require('fiber')

counter = 0
function increment(period)
    while true do
        counter = counter + 1
        fiber.sleep(period)
    end
end

increment_fiber = fiber.create(increment, 2)
require('console').start()
```

After running the script above, print the information about the fiber: a fiber ID, its status, and the counter value.

```
tarantool> print('ID: ' .. increment_fiber:id() .. '\nStatus: ' .. increment_
↪fiber:status() .. '\nCounter: ' .. counter)
ID: 104
Status: suspended
Counter: 8
---
...
```

Then, cancel the fiber and print the information about the fiber one more time. This time the fiber status is `dead`.

```
tarantool> increment_fiber:cancel()
---
...

tarantool> print('ID: ' .. increment_fiber:id() .. '\nStatus: ' .. increment_
↪fiber:status() .. '\nCounter: ' .. counter)
ID: 104
Status: dead
Counter: 12
---
...
```

`fiber.yield()`

Передача управления планировщику. Работает аналогично *fiber.sleep(0)*.

**Исключение** см. *Пример неудачной передачи управления*

**Пример:**

In the example below, two fibers are associated with the same function. Each fiber yields control after printing a greeting.

```
-- app.lua --
fiber = require('fiber')

function greet()
    while true do
        print('Enter a name:')
        name = io.read()
        print('Hello, '..name..'. I am fiber '..fiber.id())
        fiber.yield()
    end
end

for i = 1, 2 do
    fiber_object = fiber.create(greet)
    fiber_object:cancel()
end
```

The output might look as follows:

```
$ tarantool app.lua
Enter a name:
John
Hello, John. I am fiber 104
Enter a name:
Jane
Hello, Jane. I am fiber 105
```

`fiber.status(`[*fiber_object*]`)`

Return the status of the current fiber. If the `fiber_object` is passed, return the status of the specified fiber.

**Параметры**

- `fiber_object` – (optional) the fiber object

**возвращает** the status of `fiber`. One of: `dead`, `suspended`, or `running`.

**тип возвращаемого значения** строка

**Пример:**

```
tarantool> fiber.status()
---
- running
...
```

`fiber.info(`*{[backtrace/bt]}*`)`

Возврат информации о всех файберах.

**Параметры**

- `backtrace` (`boolean`) – show backtrace. Default: `true`. Set to `false` to show less information (symbol resolving can be expensive).
- `bt` (`boolean`) – same as `backtrace`, but with lower priority.

**возвращает** number of context switches (`csw`), backtrace, total memory, used memory, fiber ID (`fid`), fiber name. If fiber.top is enabled or Tarantool was built with

ENABLE_FIBER_TOP, processor time (time) is also returned.

**тип возвращаемого значения** таблица

**Return values explained**

- csw – number of context switches.
- backtrace, bt – each fiber's stack trace, showing where it originated and what functions were called.
- memory:
  - total – total memory occupied by the fiber as a C structure, its stack, etc.
  - used – actual memory used by the fiber.
- time – **duplicates the «time» entry from *fiber.top().cpu* for each fiber.** Only shown if fiber.top is enabled.

**Пример:**

```
tarantool> fiber.info({ bt = true })
---
- 101:
    csw: 1
    backtrace:
    - C: '#0  0x5dd130 in lbox_fiber_id+96'
    - C: '#1  0x5dd13d in lbox_fiber_stall+13'
    - L: stall in =[C] at line -1
    - L: (unnamed) in @builtin/fiber.lua at line 59
    - C: '#2  0x66371b in lj_BC_FUNCC+52'
    - C: '#3  0x628f28 in lua_pcall+120'
    - C: '#4  0x5e22a8 in luaT_call+24'
    - C: '#5  0x5dd1a9 in lua_fiber_run_f+89'
    - C: '#6  0x45b011 in fiber_cxx_invoke(int (*)(__va_list_tag*), __va_list_
↪tag*)+17'
    - C: '#7  0x5ff3c0 in fiber_loop+48'
    - C: '#8  0x81ecf4 in coro_init+68'
    memory:
    total: 516472
    used: 0
    time: 0
    name: lua
    fid: 101
  102:
    csw: 0
    backtrace:
    - C: '#0  (nil) in +63'
    - C: '#1  (nil) in +63'
    memory:
    total: 516472
    used: 0
    time: 0
    name: on_shutdown
    fid: 102

...
```

fiber.top()

Отображение всех активных файберов и потребляемых ими ресурсов ЦП.

**возвращает** таблица с двумя записями:cpu и cpu_misses

cpu это ещё одна таблица, в которой ключами являются строки с ID и именами файберов. Для каждого файбера доступны 3 метрики:

1. instant (in percent), which indicates the share of time the fiber was executing during the previous event loop iteration.
2. average (in percent), which is calculated as an exponential moving average of instant values over all the previous event loop iterations.
3. time (в секундах) определяет процессорное время, потраченное на обработку каждого файбера за время его существования.

   Запись time также добавляется к выводу информации о файбере с помощью fiber.info() (дублируется запись time из fiber.top().cpu для каждого файбера).

   Обратите внимание, что подсчет time ведется, только если активна функция fiber.top().

cpu_misses показывает число раз, когда поток TX регистрировал перенос файбера на другое ядро процессора во время последней итерации цикла обработки событий. fiber.top() использует счетчик меток времени для измерения времени выполнения каждого файбера. Однако в разных ядрах могут быть разные значения счетчика, поэтому полагаться на разность показаний счетчика можно только в том случае, если оба измерения были проведены на одном и том же ядре; в противном случае разность показаний может даже быть отрицательной. Когда поток TX переносится на другое ядро процессора, Tarantool просто предполагает, что разность показаний была нулевой для последнего измерения. Это снижает точность вычислений, поэтому больше значение cpu misses, тем ниже точность результатов fiber.top().

---

**Примечание:** With *2.11.0*, cpu_misses is deprecated and always returns 0.

---

**Пример:**

```
tarantool> fiber.top()
---
- cpu:
    107/lua:
      instant: 30.967324490456
      time: 0.351821993
      average: 25.582738345233
    104/lua:
      instant: 9.6473633128437
      time: 0.110869897
      average: 7.9693406131877
    101/on_shutdown:
      instant: 0
      time: 0
      average: 0
    103/lua:
      instant: 9.8026528631511
      time: 0.112641118
      average: 18.138387232255
    106/lua:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      instant: 20.071174377224
      time: 0.226901357
      average: 17.077908441831
    102/interactive:
      instant: 0
      time: 9.6858e-05
      average: 0
    105/lua:
      instant: 9.2461986412164
      time: 0.10657528
      average: 7.7068458630827
    1/sched:
      instant: 20.265286315108
      time: 0.237095335
      average: 23.141537169257
  cpu_misses: 0
...
```

Notice that by default new fibers created due to *fiber.create* are named „lua“ so it is better to set their names explicitly via *fiber_object:name(„name“)*.

There are several system fibers in `fiber.top()` output that might be useful:

- `sched` is a special system fiber. It schedules tasks to other fibers, if any, and also handles some `libev` events.

  It can have high `instant` and `average` values in `fiber.top()` output in two cases:

  – The instance has almost no load - then practically only `sched` is executing, and the other fibers are sleeping. So relative to the other fibers, `sched` may have almost 100% load.

  – `sched` handles a large number of system events. This should not cause performance problems.

- `main` fibers process requests that come over the network (iproto requests). There are several such fibers, and new ones are created if needed. When a new request comes in, a free fiber takes it and executes it. The request can be a typical `select`/`replace`/`delete`/`insert` or a function call. For example, *conn:eval()* or *conn:call()*.

---

**Примечание:** Enabling `fiber.top()` slows down fiber switching by about 15%, so it is disabled by default. To enable it, use `fiber.top_enable()`. To disable it after you finished debugging, use `fiber.top_disable()`.

---

`fiber.kill(`*id*`)`

Locate a fiber by its numeric ID and cancel it. In other words, *fiber.kill()* combines *fiber.find()* and *fiber_object:cancel()*.

**Параметры**

- `id` – the ID of the fiber to be cancelled.

**Исключение** указанный файбер отсутствует, или отмена невозможна.

**Пример:**

```
tarantool> fiber.kill(fiber.id()) -- функция с self может вызвать окончание␣
↪программы
---
- error: fiber is cancelled
...
```

fiber.testcancel()
Проверка отмены действующего файбера и выдача исключения, если файбер отменен.

---

**Примечание:** Даже при исключении файбер будет отменен. Большинство вызовов проверяют `fiber.testcancel()`. Однако некоторые функции (`id`, `status`, `join` и т.д.) не вернут ошибку. Мы рекомендуем разработчикам приложений реализовать случайные проверки *fiber.testcancel()* и максимально быстро завершить выполнение файбера, если он был отменен.

---

**Пример:**

```
tarantool> fiber.testcancel()
---
- error: fiber is cancelled
...
```

fiber.set_max_slice(*slice*)
Set the default maximum slice for all fibers. A *fiber slice* limits the time period of executing a fiber without yielding control.

**Параметры**

- `slice` (`number/table`) – a fiber slice, which can one of the following:
  - a time period (in seconds) that specifies the error slice. Example: `fiber.set_max_slice(3)`.
  - a table that specifies the warning and error slices (in seconds). Example: `fiber.set_max_slice({warn = 1.5, err = 3})`.

**Пример:**

The example below shows how to use `set_max_slice` to limit the slice for all fibers. *fiber.check_slice()* is called inside a long-running operation to determine whether a slice for the current fiber is over.

```
-- app.lua --
fiber = require('fiber')
clock = require('clock')

fiber.set_max_slice({warn = 1.5, err = 3})
time = clock.monotonic()
function long_operation()
    while clock.monotonic() - time < 5 do
        fiber.check_slice()
        -- Long-running operation  --
    end
end

long_operation_fiber = fiber.create(long_operation)
```

The output should look as follows:

```
$ tarantool app.lua
fiber has not yielded for more than 1.500 seconds
FiberSliceIsExceeded: fiber slice is exceeded
```

`fiber.set_slice(`*slice*`)`

Set a slice for the current fiber execution. A *fiber slice* limits the time period of executing a fiber without yielding control.

**Параметры**

- `slice` (`number/table`) – a fiber slice, which can one of the following:
  - a time period (in seconds) that specifies the error slice. Example: `fiber.set_slice(3)`.
  - a table that specifies the warning and error slices (in seconds). Example: `fiber.set_slice({warn = 1.5, err = 3})`.

**Пример:**

The example below shows how to use `set_slice` to limit the slice for the current fiber execution. *fiber.check_slice()* is called inside a long-running operation to determine whether a slice for the current fiber is over.

```
-- app.lua --
fiber = require('fiber')
clock = require('clock')

time = clock.monotonic()
function long_operation()
    fiber.set_slice({warn = 1.5, err = 3})
    while clock.monotonic() - time < 5 do
        fiber.check_slice()
        -- Long-running operation  --
    end
end

long_operation_fiber = fiber.create(long_operation)
```

The output should look as follows.

```
$ tarantool app.lua
fiber has not yielded for more than 1.500 seconds
FiberSliceIsExceeded: fiber slice is exceeded
```

`fiber.extend_slice(`*slice*`)`

Extend a *slice* for the current fiber execution. For example, if the default error slice is set using *fiber.set_max_slice()* to 3 seconds, `extend_slice(1)` extends the error slice to 4 seconds.

**Параметры**

- `slice` (`number/table`) – a fiber slice, which can one of the following:
  - a time period (in seconds) that specifies the error slice. Example: `fiber.extend_slice(1)`.
  - a table that specifies the warning and error slices (in seconds). Example: `fiber.extend_slice({warn = 0.5, err = 1})`.

**Пример:**

The example below shows how to use `extend_slice` to extend the slice for the current fiber execution. The default fiber slice is set using `set_max_slice`.

```
-- app.lua --
fiber = require('fiber')
clock = require('clock')

fiber.set_max_slice({warn = 1.5, err = 3})
time = clock.monotonic()
function long_operation()
    fiber.extend_slice({warn = 0.5, err = 1})
    while clock.monotonic() - time < 5 do
        fiber.check_slice()
        -- Long-running operation  --
    end
end

long_operation_fiber = fiber.create(long_operation)
```

The output should look as follows.

```
$ tarantool app.lua
fiber has not yielded for more than 2.000 seconds
FiberSliceIsExceeded: fiber slice is exceeded
```

`FiberSliceIsExceeded` is thrown after 4 seconds.

`fiber.check_slice()`

Check whether a slice for the current fiber is over. A *fiber slice* limits the time period of executing a fiber without yielding control.

**Пример:**

See the examples for the following functions:

- *fiber.set_max_slice()*
- *fiber.set_slice()*
- *fiber.extend_slice()*

`fiber.time()`

**возвращает** текущее системное время (в секундах с начала отсчета) в виде Lua-числа. Время берется из часов событийного цикла, поэтому вызов полезен лишь для создания искусственных ключей кортежа.

**тип возвращаемого значения** число

**Пример:**

```
tarantool> fiber.time(), fiber.time()
---
- 1448466279.2415
- 1448466279.2415
...
```

fiber.time64()

**возвращает** текущее системное время (в микросекундах с начала отсчета) в виде 64-битного целого числа. Время берется из часов событийного цикла.

**тип возвращаемого значения** cdata (ctype<int64_t>)

**Пример:**

```
tarantool> fiber.time(), fiber.time64()
---
- 1448466351.2708
- 1448466351270762
...
```

fiber.clock()

Получение монотонного времени в секундах. Для вычисления таймаутов лучше использовать `fiber.clock()`, поскольку *fiber.time()* сообщает системное время, а оно может меняться при изменениях в системе.

**возвращает** количество секунд в виде числа с плавающей точкой, представляющего собой время с некоторого момента в прошлом, которое гарантированно не изменится в течение всего времени процесса

**тип возвращаемого значения** число

**Пример:**

```
tarantool> start = fiber.clock()
---
...
tarantool> print(start)
248700.58805
---
...
tarantool> print(fiber.time(), fiber.time()-start)
1600785979.8291 1600537279.241
---
...
```

fiber.clock64()

То же, что и *fiber.clock()*, но в микросекундах.

**возвращает** количество секунд в виде 64-битного целого числа, представляющего собой время с некоторого момента в прошлом, которое гарантированно не изменится в течение всего времени процесса

**тип возвращаемого значения** cdata (ctype<int64_t>)

object fiber_object

fiber_object:id()

**Параметры**

- `fiber_object` – как правило, это объект, полученный в результате вызова *fiber.create*, *fiber.self* или *fiber.find*

**возвращает** ID of the fiber.

**тип возвращаемого значения** число

`fiber.self():id()` может также быть выражен как `fiber.id()`.

**Пример:**

```
tarantool> fiber_object = fiber.self()
---
...
tarantool> fiber_object:id()
---
- 101
...
```

`fiber_object:name()`

**Параметры**

- `fiber_object` – как правило, это объект, полученный в результате вызова *fiber.create*, *fiber.self* или *fiber.find*

**возвращает** имя файбера.

**тип возвращаемого значения** строка

`fiber.self():name()` может также быть выражен как `fiber.name()`.

**Пример:**

```
tarantool> fiber.self():name()
---
- interactive
...
```

`fiber_object:name(`*name*[, *options* ]`)`

Change the fiber name. By default a Tarantool server's interactive-mode fiber is named „interactive“ and new fibers created due to *fiber.create* are named „lua“. Giving fibers distinct names makes it easier to distinguish them when using *fiber.info* and *fiber.top()*. Max length is 255.

**Параметры**

- `fiber_object` – как правило, это объект, полученный в результате вызова *fiber.create*, *fiber.self* или *fiber.find*
- `name` (*string*) – новое имя файбера.
- `options` –
  - `truncate=true` – truncates the name to the max length if it is too long. If this option is false (the default), `fiber.name(new_name)` fails with an exception if a new name is too long. The name length limit is 255 (since version *2.4.1*).

**возвращает** nil

**Пример:**

```
tarantool> fiber.self():name('non-interactive')
---
...
```

`fiber_object:status()`
Возврат статуса указанного файбера.

**Параметры**

- `fiber_object` – как правило, это объект, полученный в результате вызова *fiber.create*, *fiber.self* или *fiber.find*

**возвращает** статус файбера: “dead” (недоступен), “suspended” (приостановлен) или “running” (активен).

**тип возвращаемого значения** строка

`fiber.self():status()` can also be expressed as `fiber.status()`.

**Пример:**

```
tarantool> fiber.self():status()
---
- running
...
```

`fiber_object:cancel()`
Send a cancellation request to the fiber. Running and suspended fibers can be cancelled. After a fiber has been cancelled, attempts to operate on it cause errors, for example, *fiber_object:name()* causes `error: the fiber is dead`. But a dead fiber can still report its ID and status.

Отмена файбера происходит асинхронно. Чтобы дождаться окончания отмены, используйте *fiber_object:join()*. После вызова `fiber_object:cancel()` файбер может проверить, был ли он удален. Если он этого не сделает, его отменить невозможно.

**Параметры**

- `fiber_object` – как правило, это объект, полученный в результате вызова *fiber.create*, *fiber.self* или *fiber.find*

**возвращает** nil

Возможные ошибки: нельзя отменить указанный объект файбера.

**Пример:**

See the *fiber.sleep()* example.

`fiber_object:set_max_slice(`*slice*`)`
Set a fiber's maximum slice. A *fiber slice* limits the time period of executing a fiber without yielding control.

**Параметры**

- `slice` (`number/table`) – a fiber slice, which can one of the following:
  - a time period (in seconds) that specifies the error slice. Example: `long_operation_fiber.set_max_slice(3)`.
  - a table that specifies the warning and error slices (in seconds). Example: `long_operation_fiber.set_max_slice({warn = 1.5, err = 3})`.

**Пример:**

The example below shows how to use `set_max_slice` to limit the fiber slice. *fiber.check_slice()* is called inside a long-running operation to determine whether a slice for the fiber is over.

```
-- app.lua --
fiber = require('fiber')
clock = require('clock')

time = clock.monotonic()
function long_operation()
    while clock.monotonic() - time < 5 do
        fiber.check_slice()
        -- Long-running operation  --
    end
end

long_operation_fiber = fiber.new(long_operation)
long_operation_fiber:set_max_slice({warn = 1.5, err = 3})
```

The output should look as follows.

```
$ tarantool app.lua
fiber has not yielded for more than 1.500 seconds
FiberSliceIsExceeded: fiber slice is exceeded
```

`fiber_object.storage`

A local storage within the fiber. It is a Lua table created when it is first accessed. The storage can contain any number of named values, subject to memory limitations. Naming may be done with *fiber_object*`.storage.`*name* or *fiber_object*`.storage['`*name*`'].` or with a number *fiber_object*`.storage[`*number*`]`. Values may be either numbers or strings.

`fiber.storage` is destroyed when the fiber is finished, regardless of how is it finished – via *fiber_object*`:cancel()`, or the fiber's function did „return". Moreover, the storage is cleaned up even for pooled fibers used to serve IProto requests. Pooled fibers never really die, but nonetheless their storage is cleaned up after each request. That makes possible to use `fiber.storage` as a full featured request-local storage. This behavior is implemented in versions *2.2.3*, *2.3.2*, *2.4.1*, and all later versions.

This storage may be created for a fiber, no matter how the fiber itself is created – from C or from Lua. For example, a fiber can be created in C using `fiber_new()`, then it can insert into a space, which has Lua `on_replace` triggers, and one of the triggers can create `fiber.storage`. That storage is deleted when the fiber is stopped.

**Пример:**

The example below shows how to save the last entered name in a fiber storage and get this value before cancelling a fiber.

```
-- app.lua --
fiber = require('fiber')

function greet()
    while true do
        print('Enter a name:')
        name = io.read()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        if name ~= 'bye' then
            fiber.self().storage.name = name
            print('Hello, ' .. name)
        else
            print('Goodbye, ' .. fiber.self().storage['name'])
            fiber.self():cancel()
        end
    end
end

fiber_object = fiber.create(greet)
```

The output might look as follows:

```
$ tarantool app.lua
Enter a name:
John
Hello, John
Enter a name:
Jane
Hello, Jane
Enter a name:
bye
Goodbye, Jane
```

См. также *box.session.storage*.

`fiber_object:set_joinable(`*is_joinable*`)`

Make a fiber joinable. A joinable fiber can be waited for using *fiber_object:join()*.

The best practice is to call `fiber_object:set_joinable()` before the fiber function begins to execute because otherwise the fiber could become `dead` before `fiber_object:set_joinable()` takes effect. The usual sequence could be:

1. Вызов `fiber.new()` вместо `fiber.create()` для создания нового объекта файбера fiber_object.

   Не передавать управление, поскольку это приведет к началу работы функции с файбером.

2. Вызов `fiber_object:set_joinable(true)`, чтобы сделать новый объект файбера fiber_object присоединяемым.

   Сейчас можно передать управление.

3. Вызов `fiber_object:join()`.

   Как правило, следует вызвать `fiber_object:join()`, в противном случае, статус файбера может перейти в „suspended“ (приостановлен) после выполнения функции, а не „dead“ (недоступен).

**Параметры**

- `is_joinable` (`boolean`) – the boolean value that specifies whether the fiber is joinable

**возвращает** nil

**Пример:**

See the *fiber_object.join()* example.

`fiber_object:join()`

Join a fiber. Joining a fiber enables you to get the result returned by the fiber's function.

Joining a fiber runs the fiber's function and waits until the fiber's status is `dead`. Normally a status becomes `dead` when the function execution finishes. Joining the fiber causes a yield, therefore, if the fiber is currently in the `suspended` state, execution of its fiber function resumes.

Note that joining a fiber works only if the fiber is created using *fiber.new()* and is made joinable using *fiber_object:set_joinable()*.

**возвращает** The `join` method returns two values:

- The boolean value that indicates whether the join is succeeded because the fiber's function ended normally.
- The return value of the fiber's function.

If the first value is `false`, then the `join` succeeded because the fiber's function ended abnormally and the second result has the details about the error, which one can unpack in the same way that one unpacks *a pcall result*.

**тип возвращаемого значения** boolean + result type, or boolean + struct error

**Пример:**

The example below shows how to get the result returned by the fiber's function.

```
fiber = require('fiber')

function add(a, b)
    return a + b
end

add_fiber = fiber.new(add, 5, 6)
add_fiber:set_joinable(true)
is_success, result = add_fiber:join()
print('Is successful: '.. tostring(is_success))
print('Returned value: '..result)
```

The output should look as follows.

```
$ tarantool app.lua
Is successful: true
Returned value: 11
```

### Пример неудачной передачи управления

Предупреждение: функция *yield()* и любая функция, которая неявно передает управление (например, *sleep()*), может упасть (выдать исключение).

For example, this function has a loop that repeats until *cancel()* happens. The last thing that it will print is „before yield“, which demonstrates that `yield()` failed, the loop did not continue until *testcancel()* failed.

```
fiber = require('fiber')
function function_name()
  while true do
    print('before testcancel')
    fiber.testcancel()
    print('before yield')
    fiber.yield()
  end
end
fiber_object = fiber.create(function_name)
fiber.sleep(.1)
fiber_object:cancel()
```

### Каналы

Вызов `fiber.channel()` для создания и получение нового объекта канала.

Вызов других процедур по каналу для отправки сообщений, получения сообщений или проверки статуса канала.

Message exchange is synchronous. The Lua garbage collector will mark or free the channel when no one is using it, as with any other Lua object. Use object-oriented syntax, for example, `channel:put(message)` rather than `fiber.channel.put(message)`.

`fiber.channel`(*capacity*)
Создание нового канала связи.

**Параметры**

- `capacity` (`int`) – максимальное количество слотов (спейсы для сообщений `channel:put`), которые можно использовать одновременно. По умолчанию, 0.

**возвращает** новый объект — канал.

**тип возвращаемого значения** userdata. В консоли объект сериализуется и отображается как `channel: [число]`, где `[число]` — значение, возвращаемое функцией *channel_object:count()*.

`object channel_object`

`channel_object:put`(*message*, *timeout*)
Отправка сообщения по каналу связи. Если канал заполнен, `channel:put()` ожидает, пока не освободится слот в канале.

---

**Примечание:** The default *channel capacity* is 0. With this default value, `channel:put()` *waits infinitely* until `channel:get()` is called.

---

**Параметры**

- `message` (`lua-value`) – то, что отправляется, как правило, строка, число или таблица
- `timeout` (`number`) – максимальное количество секунд ожидания, чтобы слот освободился. Значение по умолчанию: бесконечность.

**возвращает** Если указан параметр времени ожидания timeout, и в канале нет свободного слота в течение указанного времени, возвращается значение `false` (ложь). Если канал закрыт, возвращается значение `false`. В остальных случаях возвращается значение `true` (истина), которое указывает на успешную отправку.

**тип возвращаемого значения** boolean (логический)

`channel_object:close()`
Закрытие канала. Все, кто находится в режиме ожидания в канале, отключаются. Все последующие операции `channel:get()` вернут нулевое значение `nil`, а все последующие операции `channel:put()` вернут `false` (ложь).

`channel_object:get(`$\left[timeout\right]$`)`
Перехват и удаление сообщения из канала. Если канал пуст, `channel:get()` будет ожидать сообщения.

**Параметры**

- `timeout` (`number`) – максимальное количество секунд ожидания сообщения. Значение по умолчанию: бесконечность.

**возвращает** Если указан параметр времени ожидания timeout, и в канале нет сообщения в течение указанного времени, возвращается нулевое значение `nil`. Если канал закрыт, возвращается значение `nil`. В остальных случаях возвращается сообщение, отправленное на канал с помощью `channel:put()`.

**тип возвращаемого значения** как правило, строка, число или таблица, как определяет `channel:put()`

`channel_object:is_empty()`
Проверка пустоты канала (отсутствие сообщений).

**возвращает** `true` (истина), если канал пуст. В противном случае, `false` (ложь).

**тип возвращаемого значения** boolean (логический)

`channel_object:count()`
Определение количества сообщений в канале.

**возвращает** количество сообщений.

**тип возвращаемого значения** число

`channel_object:is_full()`
Проверка заполненности канала.

**возвращает** `true` (истина), если канал заполнен (количество сообщений в канале равно количеству слотов, то есть нет места для новых сообщений). В противном случае, `false` (ложь).

**тип возвращаемого значения** boolean (логический)

`channel_object:has_readers()`
Проверка пустого канала на наличие читателей в состоянии ожидания сообщения после отправки запросов `channel:get()`.

**возвращает** `true` (истина), если на канале есть читатели в ожидании сообщения. В противном случае, `false` (ложь).

**тип возвращаемого значения** boolean (логический)

`channel_object:has_writers()`
Проверка полного канала на наличие писателей в состоянии ожидания после отправки запросов `channel:put()`.

**возвращает** `true` (истина), если на канале есть писатели в состоянии ожидании. В противном случае, `false` (ложь).

**тип возвращаемого значения** boolean (логический)

`channel_object:is_closed()`

**возвращает** `true` (истина), если канал уже закрыт. В противном случае, `false` (ложь).

**тип возвращаемого значения** boolean (логический)

### Пример

В данном примере дается примерное представление о том, как должны выглядеть функции для файберов. Предполагается, что на функции ссылается *fiber.create()*.

```
fiber = require('fiber')
 channel = fiber.channel(10)
 function consumer_fiber()
     while true do
         local task = channel:get()
         ...
     end
 end

 function consumer2_fiber()
     while true do
         -- 10 секунд
         local task = channel:get(10)
         if task ~= nil then
             ...
         else
             -- время ожидания
         end
     end
 end

 function producer_fiber()
     while true do
         task = box.space...:select{...}
         ...
         if channel:is_empty() then
             -- канал пуст
         end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        if channel:is_full() then
            -- канал полон
        end

        ...
        if channel:has_readers() then
            -- есть файберы
            -- которые ожидают данные
        end
        ...

        if channel:has_writers() then
            -- есть файберы
            -- которые ожидают читателей
        end
        channel:put(task)
    end
end

function producer2_fiber()
    while true do
        task = box.space...select{...}
        -- 10 секунд
        if channel:put(task, 10) then
            ...
        else
            -- время ожидания
        end
    end
end
```

### Условные переменные

Вызов `fiber.cond()` используется для создания именованной условной переменной, которая будет называться „cond“ для примеров данного раздела.

Вызов `cond:wait()` используется, чтобы заставить файбер ожидать сигнал, с помощью условной переменной.

Вызов `cond:signal()` используется, чтобы отправить сигнал для пробуждения отдельного файбера, который выполнил запрос `cond:wait()`.

Вызов `cond:broadcast()` используется для отправки сигнала всем файберам, которые выполнили `cond:wait()`.

`fiber.cond()`
: Создание новой условной переменной.

  **возвращает** новая условная переменная.

  **тип возвращаемого значения** Lua-объект

```
object cond_object
```

`cond_object:wait([timeout])`
Перевод файбера в режим ожидания до пробуждения другим файбером с помощью метода `signal()` или `broadcast()`. Переход в режим ожидания вызывает неявную передачу управления *fiber.yield()*.

**Параметры**

- `timeout` – количество секунд ожидания, по умолчанию = всегда.

**возвращает** Если указан параметр времени ожидания timeout, и сигнал не передается в течение указанного времени, `wait()` вернет значение false (ложь). Если передается `signal()` или `broadcast()`, `wait()` вернет true (истина).

**тип возвращаемого значения** boolean (логический)

`cond_object:signal()`
Пробуждение отдельного файбера, который выполнил `wait()` для той же переменной. Не выполняет передачу управления (yield).

**тип возвращаемого значения** nil

`cond_object:broadcast()`
Пробуждение всех файберов, которые выполнили `wait()` для той же переменной. Не выполняет передачу управления (yield).

**тип возвращаемого значения** nil

### Пример

Assume that a Tarantool instance is running and listening for connections on localhost port 3301. Assume that guest users have privileges to connect. We will use the *tt* utility to start two clients.

В первом терминале введите:

```
$ tt connect localhost:3301
tarantool> fiber = require('fiber')
tarantool> cond = fiber.cond()
tarantool> cond:wait()
```

Задача повиснет, поскольку `cond:wait()` – без дополнительного аргумента времени ожидания timeout – уйдет в режим ожидания до изменения условной переменной.

Во втором терминале введите:

```
$ tt connect localhost:3301
tarantool> cond:signal()
```

Теперь снова взгляните на терминал №1. Он покажет, что ожидание прекратилось, и функция `cond:wait()` вернула значение `true`.

В данном примере показана зависимость от использования глобальной условной переменной с произвольным именем `cond`. В реальной жизни разработчики следят за использованием различных имен для условных переменных в разных приложениях.

### 9.5.15 Модуль fio

#### Общие сведения

Tarantool поддерживает файловый ввод-вывод с помощью API, который аналогичен системным вызовам POSIX. Все операции проводятся асинхронно. Несколько файберов могут получать доступ к одному файлу одновременно.

Модуль `fio` включает в себя:

- функции для *стандартных действий с путем к файлу*,
- функции для *проверки наличия и типа директории или файла*,
- функции для *стандартных действий с файлами*, а также
- *постоянные*., которые совпадают с флаговыми значениями POSIX (например, `fio.c.flag.O_RDONLY` = POSIX O_RDONLY).

#### Указатель

Ниже приведен перечень всех функций и элементов модуля `fio`.

| Имя | Назначение |
|---|---|
| *fio.pathjoin()* | Формирование пути к файлу из одной или более частей строки |
| *fio.basename()* | Получение имени файла |
| *fio.dirname()* | Получение имени директории |
| *fio.abspath()* | Получение имен директории и файла |
| *fio.path.exists()* | Проверка наличия файла или директории |
| *fio.path.is_dir()* | Проверка, является ли файл или директория директорией |
| *fio.path.is_file()* | Проверка, является ли файл или директория файлом |
| *fio.path.is_link()* | Проверка, является ли файл или директория ссылкой |
| *fio.path.lexists()* | Проверка наличия файла или директории |
| *fio.umask()* | Определение битов маски |
| *fio.lstat() fio.stat()* | Получение информации об объекте файла |
| *fio.mkdir() fio.rmdir()* | Создание или удаление директории |
| *fio.chdir()* | Изменение рабочей директории |
| *fio.listdir()* | Вывод списка файлов в директории |
| *fio.glob()* | Получение файлов, имена которых совпадают с заданной строкой |
| *fio.tempdir()* | Получение имени директории для хранения временных файлов |
| *fio.cwd()* | Получение имени текущей рабочей директории |
| *fio.copytree() fio.mktree() fio.rmtree()* | Создание и удаление директорий |
| *fio.link() fio.symlink() fio.readlink() fio.unlink()* | Создание и удаление ссылок |
| *fio.rename()* | Переименование файла или директории |
| *fio.utime()* | Изменение времени обновления файла |
| *fio.copyfile()* | Копирование файла |

continues on next page

Таблица 5 – продолжение с предыдущей страницы

| Имя | Назначение |
|---|---|
| *fio.chown() fio.chmod()* | Управление правами на использование и правами владения объектами файла |
| *fio.truncate()* | Уменьшение размера файла |
| *fio.sync()* | Проверка записи изменений на диск |
| *fio.open()* | Открытие файла |
| *file-handle:close()* | Закрытие файла |
| *file-handle:pread() file-handle:pwrite()* | Чтение или запись в файл с произвольным доступом |
| *file-handle:read() file-handle:write()* | Чтение или запись в файл не с произвольным доступом |
| *file-handle:truncate()* | Изменение размера открытого файла |
| *file-handle:seek()* | Изменение позиции в файле |
| *file-handle:stat()* | Получение статистики об открытом файле |
| *file-handle:fsync() file-handle:fdatasync()* | Проверка записи изменений в открытом файле на диск |
| *fio.c* | Таблица переменных, аналогичных флаговым значениям POSIX |

### Стандартные действия с путем к файлу

`fio.pathjoin`(*partial-string*[, *partial-string ...*])

Конкатенация частей строки, разделенных „/“ для формирования пути к файлу.

**Параметры**

- `partial-string` (*string*) – одна или более строк для конкатенации.

**возвращает** путь к файлу

**тип возвращаемого значения** строка

**Пример:**

```
tarantool> fio.pathjoin('/etc', 'default', 'myfile')
---
- /etc/default/myfile
...
```

`fio.basename`(*path-name*[, *suffix*])

Удаление из полного пути к файлу всего, за исключением последней части (имени файла). Также удаление суффикса, если он передается.

Note that the basename of a path with a trailing slash is an empty string. It is different from how the Unix `basename` program interprets such a path.

**Параметры**

- `path-name` (*string*) – путь к файлу
- `suffix` (*string*) – суффикс

**возвращает** имя файла

**тип возвращаемого значения** строка

**Пример:**

```
tarantool> fio.basename('/path/to/my.lua', '.lua')
---
- my
...
```

**Example with a trailing slash:**

```
tarantool> fio.basename('/path/to/')
---
-
...
```

`fio.dirname`(*path-name*)

Удаление последней части (имени файла) из полного пути к файлу.

**Параметры**

- `path-name` (*string*) – путь к файлу

**возвращает** имя директории, то есть путь к файлу без имени файла.

**тип возвращаемого значения** строка

**Пример:**

```
tarantool> fio.dirname('/path/to/my.lua')
---
- '/path/to/'
```

`fio.abspath`(*file-name*)

Возврат полного пути к файлу на основании последней части (имени файла).

**Параметры**

- `file-name` (*string*) – имя файла

**возвращает** имя директории, то есть путь к файлу с именем файла.

**тип возвращаемого значения** строка

**Пример:**

```
tarantool> fio.abspath('my.lua')
---
- '/path/to/my.lua'
...
```

### Проверка наличия и типа директории или файла

Functions in this section are similar to some Python os.path functions.

`fio.path.exists`(*path-name*)

**Параметры**

- `path-name` (*string*) – путь к директории или файлу.

**возвращает** true (правда), если путь к файлу ссылается на директорию или файл, которые присутствуют в системе, и не представляет собой нерабочую символьную ссылку; в противном случае, false (ложь)

**тип возвращаемого значения** boolean (логический)

`fio.path.is_dir(`*path-name*`)`

**Параметры**

- `path-name` (*string*) – путь к директории или файлу.

**возвращает** true (правда), если путь к файлу ссылается на директорию; в противном случае, false (ложь)

**тип возвращаемого значения** boolean (логический)

`fio.path.is_file(`*path-name*`)`

**Параметры**

- `path-name` (*string*) – путь к директории или файлу.

**возвращает** true (правда), если путь к файлу ссылается на файл; в противном случае, false (ложь)

**тип возвращаемого значения** boolean (логический)

`fio.path.is_link(`*path-name*`)`

**Параметры**

- `path-name` (*string*) – путь к директории или файлу.

**возвращает** true (правда), если путь к файлу ссылается на символьную ссылку; в противном случае, false (ложь)

**тип возвращаемого значения** boolean (логический)

`fio.path.lexists(`*path-name*`)`

**Параметры**

- `path-name` (*string*) – путь к директории или файлу.

**возвращает** true (правда), если путь к файлу ссылается на директорию или файл, которые присутствуют в системе, и представляет собой нерабочую символьную ссылку; в противном случае, false (ложь)

**тип возвращаемого значения** boolean (логический)

### Стандартные действия с файлом

`fio.umask`(*mask-bits*)

Определение битов маски при создании файлов или директорий. Для получения более подробного описания введите `man 2 umask`.

**Параметры**

- `mask-bits` (`number`) – биты маски.

**возвращает** предыдущие биты маски.

**тип возвращаемого значения** число

**Пример:**

```
tarantool> fio.umask(tonumber('755', 8))
---
- 493
...
```

`fio.lstat`(*path-name*)
`fio.stat`(*path-name*)

Возврат информации об объекте файла. Для получения более подробной информации введите `man 2 lstat` или `man 2 stat`.

**Параметры**

- `path-name` (*string*) – путь к файлу.

**возвращает** (Если ошибки нет) таблица с полями, которые описывают размер блока файла, время создания, размер и прочие атрибуты. (В случае ошибки) возвращаются два значения: null, сообщение об ошибке.

**тип возвращаемого значения** таблица.

Кроме того, результат `fio.stat('имя-файла')` будет включать в себя методы, которые аналогичны макросам в POSIX:

- `is_blk()` = макрос S_ISBLK в POSIX,
- `is_chr()` = макрос S_ISCHR в POSIX
- `is_dir()` = макрос S_ISDIR в POSIX,
- `is_fifo()` = макрос S_ISFIFO в POSIX,
- `is_link()` = макрос S_ISLINK в POSIX,
- `is_reg()` = макрос S_ISREG в POSIX,
- `is_sock()` = макрос S_ISSOCK в POSIX.

Например, `fio.stat('/'):is_dir()` вернет true.

**Пример:**

```
tarantool> fio.lstat('/etc')
---
- inode: 1048577
  rdev: 0
  size: 12288
  atime: 1421340698
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  mode: 16877
  mtime: 1424615337
  nlink: 160
  uid: 0
  blksize: 4096
  gid: 0
  ctime: 1424615337
  dev: 2049
  blocks: 24
...
```

fio.mkdir(*path-name*[, *mode*])
fio.rmdir(*path-name*)

Создание или удаление директории. Для получения подробной информации введите `man 2 mkdir` или `man 2 rmdir`.

**Параметры**

- `path-name` (*string*) – путь к директории.
- `mode` (`number`) – Биты режима работы можно передать в виде числа или строковых постоянных, например `S_IWUSR`. Биты режима работы можно комбинировать путем обрамления их в фигурные скобки.

**возвращает** (Если ошибки нет) true. (В случае ошибки) возвращаются два значения: false, сообщение об ошибке.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.mkdir('/etc')
---
- false
...
```

fio.chdir(*path-name*)

Изменение рабочей директории. Для получения более подробной информации введите `man 2 chdir`.

**Параметры**

- `path-name` (*string*) – путь к директории.

**возвращает** (Если выполнено) true. (Если не выполнено) false.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.chdir('/etc')
---
- true
...
```

fio.listdir(*path-name*)

Вывод списка файлов в директории. Результат аналогичен результату выполнения команды `ls` в терминале.

**Параметры**

- `path-name` (*string*) – путь к директории.

**возвращает** (Если ошибки нет) список файлов. (В случае ошибки) возвращаются два значения: null, сообщение об ошибке.

**тип возвращаемого значения** таблица

**Пример:**

```
tarantool> fio.listdir('/usr/lib/tarantool')
---
- - mysql
...
```

`fio.glob`(*path-name*)

Возврат списка файлов, имена которых совпадают с введенной строкой. Список составляется с одним флагом, который контролирует поведение функции: `GLOB_NOESCAPE`. Для получения подробной информации введите `man 3 glob`.

**Параметры**

- `path-name` (*string*) – путь к файлу, который может содержать специальные символы.

**возвращает** список файлов, имена которых совпадают с введенной строкой.

**тип возвращаемого значения** таблица

**Возможные ошибки:** nil.

**Пример:**

```
tarantool> fio.glob('/etc/x*')
---
- - /etc/xdg
  - /etc/xml
  - /etc/xul-ext
...
```

`fio.tempdir()`

Возврат имени директории, которую можно использовать для хранения временных файлов.

**Пример:**

```
tarantool> fio.tempdir()
---
- /tmp/lG31e7
...
```

`fio.tempdir()` stores the created temporary directory into `/tmp` by default. Since version *2.4.1*, this can be changed by setting the `TMPDIR` environment variable. Before starting Tarantool, or at runtime by `os.setenv()`.

**Пример:**

```
tarantool> fio.tempdir()
---
- /tmp/lG31e7
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> fio.mkdir('./mytmp')
---
- true
...

tarantool> os.setenv('TMPDIR', './mytmp')
---
...

tarantool> fio.tempdir()
---
- ./mytmp/506Z0b
...
```

`fio.cwd()`
Возврат имени текущей рабочей директории.

**Пример:**

```
tarantool> fio.cwd()
---
- /home/username/tarantool_sandbox
...
```

`fio.copytree(`*from-path*, *to-path*`)`
Копирование всего из директории from-path, включая поддиректории, в to-path. Результат аналогичен результату выполнения команды `cp -r` в терминале. Директория to-path не должна быть пустой.

**Параметры**

- `from-path` (*string*) – путь.
- `to-path` (*string*) – путь.

**возвращает** (Если ошибки нет) true. (В случае ошибки) возвращаются два значения: false, сообщение об ошибке.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.copytree('/home/original','/home/archives')
---
- true
...
```

`fio.mktree(`*path-name*`)`
Создание пути, включая поддиректории, но без содержимого файла. Результат аналогичен результату выполнения команды `mkdir -p` в терминале.

**Параметры**

- `path-name` (*string*) – путь.

**возвращает** (Если ошибки нет) true. (В случае ошибки) возвращаются два значения: false, сообщение об ошибке.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.mktree('/home/archives')
---
- true
...
```

fio.rmtree(*path-name*)

Удаление указанной директории, включая поддиректории. Результат аналогичен результату выполнения команды `rm -rf` в терминале.

**Параметры**

- `path-name` (*string*) – путь.

**возвращает** (Если ошибки нет) true. (В случае ошибки) возвращаются два значения: null, сообщение об ошибке.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.rmtree('/home/archives')
---
- true
...
```

fio.link(*src*, *dst*)
fio.symlink(*src*, *dst*)
fio.readlink(*src*)
fio.unlink(*src*)

Функции для создания и удаления ссылок. Для получения подробной информации введите `man readlink`, `man 2 link`, `man 2 symlink`, `man 2 unlink`.

**Параметры**

- `src` (*string*) – имя существующего файла.
- `dst` (*string*) – связанное имя.

**возвращает** (Если ошибки нет) `fio.link`, `fio.symlink` и `fio.unlink` возвращают true (правда), `fio.readlink` возвращает ссылку. (В случае ошибки) возвращаются два значения: false|null, сообщение об ошибке.

**Пример:**

```
tarantool> fio.link('/home/username/tmp.txt', '/home/username/tmp.txt2')
---
- true
...
tarantool> fio.unlink('/home/username/tmp.txt2')
---
- true
...
```

fio.rename(*path-name*, *new-path-name*)

Переименование файла или директории. Для получения подробной информации введите `man 2 rename`.

**Параметры**

- `path-name` (*string*) – первоначальное имя.

- `new-path-name` (*string*) – новое имя.

**возвращает** (Если ошибки нет) true. (В случае ошибки) возвращаются два значения: false, сообщение об ошибке.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.rename('/home/username/tmp.txt', '/home/username/tmp.txt2')
---
- true
...
```

`fio.utime`(*file-name*[, *accesstime*[, *updatetime*]])

Change the access time and possibly also change the update time of a file. For details type `man 2 utime`. Times should be expressed as number of seconds since the epoch.

**Параметры**

- `file-name` (*string*) – имя.
- `accesstime` (`number`) – time of last access. default current time.
- `updatetime` (`number`) – time of last update. default = access time.

**возвращает** (Если ошибки нет) true. (В случае ошибки) возвращаются два значения: false, сообщение об ошибке.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.utime('/home/username/tmp.txt')
---
- true
...
```

`fio.copyfile`(*path-name*, *new-path-name*)

Копирование файла. Результат аналогичен результату выполнения команды `cp` в терминале.

**Параметры**

- `path-name` (*string*) – путь к первоначальному файлу.
- `new-path-name` (*string*) – путь к новому файлу.

**возвращает** (Если ошибки нет) true. (В случае ошибки) возвращаются два значения: false, сообщение об ошибке.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.copyfile('/home/user/tmp.txt', '/home/usern/tmp.txt2')
---
- true
...
```

`fio.chown`(*path-name*, *owner-user*, *owner-group*)
`fio.chmod`(*path-name*, *new-rights*)

Управление правами на использование и правами владения объектами файла. Для получения подробной информации введите `man 2 chown` или `man 2 chmod`.

**Параметры**

- `owner-user` (*string*) – новый UID пользователя.
- `owner-group` (*string*) – новый UID группы.
- `new-rights` (`number`) – новые права

**возвращает** null

**Пример:**

```
tarantool> fio.chmod('/home/username/tmp.txt', tonumber('0755', 8))
---
- true
...
tarantool> fio.chown('/home/username/tmp.txt', 'username', 'username')
---
- true
...
```

`fio.truncate`(*path-name*, *new-size*)

Уменьшение размера файла до указанного значения. Для получения подробной информации введите `man 2 truncate`.

**Параметры**

- `path-name` (*string*) –
- `new-size` (`number`) –

**возвращает** (Если ошибки нет) true. (В случае ошибки) возвращаются два значения: false, сообщение об ошибке.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.truncate('/home/username/tmp.txt', 99999)
---
- true
...
```

`fio.sync`()

Проверка записи изменений на диск. Для получения подробной информации введите `man 2 sync`.

**возвращает** true – если выполнено, false – если не выполнено.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fio.sync()
---
- true
...
```

`fio.open`(*path-name*[, *flags*[, *mode*]])

Открытие файла в процессе подготовки к чтению, записи или поиску.

**Параметры**

- `path-name` (*string*) – Полный путь к открываемому файлу.

- `flags` (`number`) – Флаги могут передаваться в виде числа или в виде строковых постоянных, например „`O_RDONLY`“, „`O_WRONLY`“, „`O_RDWR`“. Флаги можно комбинировать путем обрамления их в фигурные скобки. Все флаги в Linux, как описано на странице руководства по Linux, представлены ниже:
  - O_APPEND (открывать файл в режиме добавления),
  - O_ASYNC (включать ввод-вывод, управляемый сигналом),
  - O_CLOEXEC (устанавливать флаг, связанный с закрытием),
  - O_CREAT (создать файл, если он не существует),
  - O_DIRECT (минимизировать или отключать кэширование),
  - O_DIRECTORY (завершать вызов с ошибкой, если путь не является директорией),
  - O_EXCL (завершать вызов с ошибкой, если файл не может быть создан),
  - O_LARGEFILE (открывать 64-битные файлы),
  - O_NOATIME (не обновлять время последнего доступа к файлу),
  - O_NOCTTY (не делать терминальное устройство управляющим терминалом tty),
  - O_NOFOLLOW (не открывать символьные ссылки),
  - O_NONBLOCK (открывать в неблокирующем режиме),
  - O_PATH (получить путь для низкоуровневого использования),
  - O_SYNC (включить принудительную запись, если возможно),
  - O_TMPFILE (создать безымянный временный файл),
  - O_TRUNC (урезать)

  ... и всегда используется один из флагов:

  - O_RDONLY (только для чтения),
  - O_WRONLY (только для записи) или
  - O_RDWR (для чтения и записи).
- `mode` (`number`) – Биты режима работы можно передать в виде числа или строковых постоянных, например `S_IWUSR`. Биты режима работы имеют значение, если указаны флаги `O_CREAT` или `O_TMPFILE`. Биты режима работы можно комбинировать путем обрамления их в фигурные скобки.

**возвращает** (Если ошибки нет) дескриптор файла (далее сокращенно „fh“). (В случае ошибки) возвращаются два значения: null, сообщение об ошибке.

**тип возвращаемого значения** пользовательские данные

**Возможные ошибки:** nil.

Note that since version *2.4.1* `fio.open()` returns a descriptor which can be closed manually by calling the `:close()` method, or it will be closed automatically when it has no references, and the garbage collector deletes it.

Имейте в виду, что количество файловых дескрипторов ограничено и они могут быть исчерпаны раньше, чем будет запущен сборщик мусора для сбора неиспользуемых дескрипторов. Всегда рекомендуется закрывать их вручную как можно скорее.

**Пример 1:**

```
tarantool> fh = fio.open('/home/username/tmp.txt', {'O_RDWR', 'O_APPEND'})
---
...
tarantool> fh -- отображение дескриптора файла, который возвращает fio.open
---
- fh: 11
...
```

**Пример 2:**

Using `fio.open()` with `tonumber('N', 8)` to set permissions as an octal number:

```
tarantool> fio.open('x.txt', {'O_WRONLY', 'O_CREAT'}, tonumber('644',8))
 ---
 - fh: 12
 ...
```

`object file-handle`

`file-handle:close()`

Закрытие файла, который был открыт с помощью `fio.open`. Для получения подробной информации введите `man 2 close`.

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.

**возвращает** true – если выполнено, false – если не выполнено.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fh:close() -- где fh = дескриптор файла
---
- true
...
```

`file-handle:pread(`*count*, *offset*`)`
`file-handle:pread(`*buffer*, *count*, *offset*`)`

Чтение файла с произвольным доступом независимо от текущего положения в поиске. Для получения подробной информации введите `man 2 pread`.

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.
- `buffer` – куда считать (если формат – `pread(buffer, count, offset)`)
- `count` (`number`) – количество байтов для чтения
- `offset` (`number`) – смещение в файле – где начинается чтение

Если формат – `pread(count, offset)`, возвращается строка с данными, прочитанными из файла, либо пустая строка, если не выполнено.

Если формат – `pread(buffer, count, offset)`, возвращаются данные в буфер. Буферы можно ввести с помощью *buffer.ibuf*.

**Пример:**

```
tarantool> fh:pread(25, 25)
---
- |
  elete from t8//
  insert in
...
```

file-handle:pwrite(*new-string*, *offset*)
file-handle:pwrite(*buffer*, *count*, *offset*)

Запись в файл с произвольным доступом независимо от текущего положения в поиске. Для получения подробной информации введите `man 2 pwrite`.

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.
- `new-string` (*`string`*) – записываемое значение (если формат – `pwrite(new-string, offset)`)
- `buffer` (`cdata`) – записываемое значение (если формат – `pwrite(buffer, count, offset)`)
- `count` (`number`) – количество байтов для чтения
- `offset` (`number`) – смещение в файле – где начинается запись

**возвращает** true – если выполнено, false – если не выполнено.

**тип возвращаемого значения** boolean (логический)

Если формат –`pwrite(new-string, offset)`, строка записывается в файл до конца строки.

Если формат – `pwrite(buffer, count, offset)`, содержимое буфера записывается в файл в объеме, указанном в `count`. Буферы можно ввести с помощью *buffer.ibuf*.

```
tarantool> ibuf = require('buffer').ibuf()
---
...

tarantool> fh:pwrite(ibuf, 1, 0)
---
- true
...
```

file-handle:read(⌊*count*⌋)
file-handle:read(*buffer*, *count*)

Чтение файла не с произвольным доступом. Для получения подробной информации введите `man 2 read` или `man 2 write`.

---

**Примечание:** `fh:read` и `fh:write` влияют на положение поиска по файлу, и это следует учитывать при работе нескольких файберов над одним файлом. Существует возможность ограничения или запрета на доступ к файлу с помощью `fiber.ipc`. Можно ограничить или запретить доступ к файлам из других файберов с помощью *fiber.cond()* или *fiber.channel()*.

---

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.
- `buffer` – куда считать (если формат – `read(buffer, count)`)
- `count` (`number`) – количество байтов для чтения

**возвращает**

- Если формат – `read()` – без `count` – считываются все байты в файле.
- Если формат – `read()` или `read([count])`, возвращается строка с данными, прочитанными из файла, либо пустая строка, если не выполнено.
- Если формат – `read(buffer, count)`, возвращаются данные в буфер. Буферы можно ввести с помощью *buffer.ibuf*.
- В случае ошибки метод возвращает `nil, err` и устанавливает ошибку на `errno`.

```
tarantool> ibuf = require('buffer').ibuf()
---
...

tarantool> fh:read(ibuf:reserve(5), 5)
---
- 5
...

tarantool> require('ffi').string(ibuf:alloc(5),5)
---
- abcde
```

`file-handle:write(`*new-string*`)`
`file-handle:write(`*buffer*, *count*`)`

Запись в файл не с произвольным доступом. Для получения подробной информации введите `man 2 write`.

---

**Примечание:** `fh:read` и `fh:write` влияют на положение поиска по файлу, и это следует учитывать при работе нескольких файберов над одним файлом. Существует возможность ограничения или запрета на доступ к файлу с помощью `fiber.ipc`. Можно ограничить или запретить доступ к файлам из других файберов с помощью *fiber.cond()* или *fiber.channel()*.

---

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.
- `new-string` (*`string`*) – записываемое значение (если формат – `write(new-string)`)
- `buffer` (`cdata`) – записываемое значение (если формат – `write(buffer, count)`)
- `count` (`number`) – количество байтов для чтения

**возвращает** true – если выполнено, false – если не выполнено.

**тип возвращаемого значения** boolean (логический)

Если формат – `write(new-string)`, строка записывается в файл до конца строки.

Если формат – `write(buffer, count)`, содержимое буфера записывается в файл в объеме, указанном в `count`. Буферы можно ввести с помощью *buffer.ibuf*.

**Пример:**

```
tarantool> fh:write("new data")
---
- true
...
tarantool> ibuf = require('buffer').ibuf()
---
...
tarantool> fh:write(ibuf, 1)
---
- true
...
```

`file-handle:truncate(`*new-size*`)`

Изменение размера открытого файла. Отличается от функции `fio.truncate`, которая изменяет размер закрытого файла.

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.

**возвращает** true – если выполнено, false – если не выполнено.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> fh:truncate(0)
---
- true
...
```

`file-handle:seek(`*position*`[`, *offset-from*`])`

Изменение положения в файле на указанное. Для получения подробной информации введите `man 2 seek`.

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.
- `position` (`number`) – искомое положение
- `offset-from` (*string*) – „SEEK_END“ = конец файла, „SEEK_CUR“ = текущее положение, „SEEK_SET“ = начало файла.

**возвращает** новое положение, если выполнено

**тип возвращаемого значения** число

**Возможные ошибки:** nil.

**Пример:**

```
tarantool> fh:seek(20, 'SEEK_SET')
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- 20
...
```

file-handle:stat()

Возврат статистики об открытом файле. Отличается от функции `fio.stat`, которая возвращает статистику о закрытом файле. Для получения подробной информации введите `man 2 stat`.

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.

**возвращает** подробная информация о файле.

**тип возвращаемого значения** таблица

**Пример:**

```
tarantool> fh:stat()
---
- inode: 729866
  rdev: 0
  size: 100
  atime: 140942855
  mode: 33261
  mtime: 1409430660
  nlink: 1
  uid: 1000
  blksize: 4096
  gid: 1000
  ctime: 1409430660
  dev: 2049
  blocks: 8
...
```

file-handle:fsync()
file-handle:fdatasync()

Проверка записи изменений в открытом файле на диск. Ср. с `fio.sync` для всех файлов. Для получения подробной информации введите `man 2 fsync` or `man 2 fdatasync`.

**Параметры**

- `fh` (`userdata`) – дескриптор файла, который возвращает `fio.open()`.

**возвращает** true – если выполнено, false – если не выполнено.

**Пример:**

```
tarantool> fh:fsync()
---
- true
...
```

#### Постоянные для файлового ввода-вывода

`fio.c`
: Таблица с постоянными, которые совпадают с флаговыми значениями в POSIX на целевой платформе (см. `man 2 stat`).

**Пример:**

```
tarantool> fio.c
---
- seek:
    SEEK_SET: 0
    SEEK_END: 2
    SEEK_CUR: 1
  mode:
    S_IWGRP: 16
    S_IXGRP: 8
    S_IROTH: 4
    S_IXOTH: 1
    S_IRUSR: 256
    S_IXUSR: 64
    S_IRWXU: 448
    S_IRWXG: 56
    S_IWOTH: 2
    S_IRWXO: 7
    S_IWUSR: 128
    S_IRGRP: 32
  flag:
    O_EXCL: 2048
    O_NONBLOCK: 4
    O_RDONLY: 0
    <...>
...
```

### 9.5.16 Модуль fun

Luafun, также известная как библиотека для функционального программирования в Lua, пользуется преимуществами LuaJIT, чтобы помочь пользователям создавать сложные функции. Модуль включает в себя «последовательные процессоры», такие как `map`, `filter`, `reduce`, `zip` – они берут написанную пользователем функцию в качестве аргумента и применяют ее к каждому элементу в последовательности, что может работать быстрее или более удобно, чем написанный пользователем цикл. Модуль включает в себя «генераторы», такие как `range`, `tabulate` и `rands` – они возвращают ограниченный или неограниченный ряд значений. Модуль включает в себя «преобразователи», «фильтры», «компоновщики» ... или, коротко говоря, все важные функции из таких языков, как Standard ML, Haskell или Erlang.

The full documentation is On the luafun section of github. However, the first chapter can be skipped because installation is already done, it's inside Tarantool. All that is needed is the usual `require` request. After that, all the operations described in the Lua fun manual will work, provided they are preceded by the name returned by the `require` request. For example:

```
tarantool> fun = require('fun')
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> for _k, a in fun.range(3) do
         >   print(a)
         > end
1
2
3
---
...
```

### 9.5.17 Модуль http

The `http` module, specifically the `http.client` submodule, provides the functionality of an HTTP client with support for HTTPS and keepalive. The HTTP client uses the libcurl library under the hood and takes into account the environment variables libcurl understands.

#### HTTP client instance

##### Default client

The `http.client` submodule provides the default HTTP client instance:

```
local http_client = require('http.client')
```

In this case, you need to make requests using the dot syntax, for example:

```
local response = http_client.get('https://httpbin.org/get')
```

#### Creating a client

If you need to configure specific HTTP client options, use the *http.client.new()* function to create the client instance:

```
local http_client = require('http.client').new()
```

In this case, you need to make requests using the colon syntax, for example:

```
local response = http_client:get('https://httpbin.org/get')
```

All the examples in this section use the HTTP client created using `http.client.new()`.

### Making requests

The *client instance* enables you to make HTTP requests.

### HTTP method

The main way of making HTTP requests is the *request* method, which accepts the following arguments:

- An HTTP method, such as `GET`, `POST`, `PUT`, and so on.
- A request URL. You can use the *uri* module to construct a URL from its components.
- (Optional) a request body for the `POST`, `PUT`, and `PATCH` methods.
- (Optional) request options, such as request headers, SSL settings, and so on.

The example below shows how to make the `GET` request to the `https://httpbin.org/get` URL:

```
local http_client = require('http.client').new()
local response = http_client:request('GET', 'https://httpbin.org/get')
```

In addition to `request`, the HTTP client provides the API for particular HTTP methods: *get*, *post*, *put*, and so on. For example, you can replace the request above by calling `get` as follows:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/get')
```

### Query parameters

To add query string parameters, use the *params* option exposed by the *request_ options* object:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/get', {
    params = { page = 1 },
})
print('URL: '..response.url)
```

In the example above, the requested URL is `https://httpbin.org/get?page=1`.

---

**Примечание:** If a parameter name or value contains a reserved character (for example, `&` or `=`), the HTTP client encodes a query string.

---

### Headers

To add headers to the request, use the *headers* option:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/headers', {
    headers = {
        ['User-Agent'] = 'Tarantool HTTP client',
        ['Authorization'] = 'Bearer abc123'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    }
})
print('Authorization: '..response:decode()['headers']['Authorization'])
```

### Cookies

You can add cookies to the request using the *headers* option:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/cookies', {
    headers = {
        ['Cookie'] = 'session_id=abc123; csrftoken=u32t4o;',
    }
})
print(response.body)
```

To learn how to obtain cookies passed in the `Set-Cookie` response header, see *Response cookies*.

### Body

#### Serialization

The HTTP client automatically serializes the content in a specific format when sending a request based on the specified `Content-Type` header. By default, the client uses the `application/json` content type and sends data serialized as JSON:

```
local http_client = require('http.client').new()
local response = http_client:post('https://httpbin.org/anything', {
    user_id = 123,
    user_name = 'John Smith'
})
print('Posted data: '..response:decode()['data'])
```

The body for the request above might look like this:

```
{
    "user_id": 123,
    "user_name": "John Smith"
}
```

To send data in the YAML or MsgPack format, set the `Content-Type` header explicitly to `application/yaml` or `application/msgpack`, for example:

```
local http_client = require('http.client').new()
local response = http_client:post('https://httpbin.org/anything', {
    user_id = 123,
    user_name = 'John Smith'
}, {
    headers = {
        ['Content-Type'] = 'application/yaml',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    }
})
print('Posted data:\n'..response:decode()['data'])
```

In this case, the request body is serialized to YAML:

```
user_id: 123
user_name: John Smith
```

### Form parameters

To send form parameters using the `application/x-www-form-urlencoded` type, use the *params* option:

```
local http_client = require('http.client').new()
local response = http_client:post('https://httpbin.org/anything', nil, {
    params = { user_id = 123, user_name = 'John Smith' },
})
print('User ID: '..response:decode()['form']['user_id'])
```

### Streaming upload

The HTTP client supports chunked writing of request data. This can be achieved as follows:

1. Set the *chunked* option to `true`. In this case, a request method returns *io_object* instead of *response_object*.
2. Use the *io_object.write()* method to write a chunk of data.
3. Call the *io_object.finish()* method to finish writing data and make a request.

The example below shows how to upload data in two chunks:

```
local http_client = require('http.client').new()
local json = require('json')

local io = http_client:post('https://httpbin.org/anything', nil, {chunked = true})
io:write('Data part 1')
io:write('Data part 2')
io:finish()
response = io:read('\r\n')
decoded_data = json.decode(response)
print('Posted data: '..decoded_data['data'])
```

#### Receiving responses

All methods that are used to *make an HTTP request* (`request`, `get`, `post`, etc.) receive *response_object*. `response_object` exposes the API required to get a response body and obtain response parameters, such as a status code, headers, and so on.

#### Status code

To get a response's status code and text, use the *response_object.status* and *response_object.reason* options, respectively:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/get')
print('Status: '..response.status..' '.. response.reason)
```

#### Headers

The *response_object.headers* option returns a set of response headers. The example below shows how to obtain the `ETag` header value:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/etag/7c876b7e')
print('ETag header value: '..response.headers['etag'])
```

#### Cookies

To obtain response cookies, use *response_object.cookies*. This option returns a Lua table where a cookie name is the key. The value is an array of two elements where the first one is the cookie value and the second one is an array with the cookie's options.

The example below shows how to obtain the `session_id` cookie value:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/cookies/set?session_id=abc123&
↪csrftoken=u32t4o&', {follow_location = false})
print("'session_id' cookie value: "..response.cookies['session_id'][1])
```

#### Response body

#### Deserialization

The HTTP client can deserialize response data to a Lua object based on the `Content-Type` response header value. To deserialize data, call the *response_object.decode()* method. In the example below, the JSON response is deserialized into a Lua object:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/json')
local document = response:decode()
print("'title' value: "..document['slideshow']['title'])
```

The following content types are supported out of the box:

- `application/json`
- `application/msgpack`
- `application/yaml`

If the response doesn't have the `Content-Type` header, the client uses `application/json`.

To deserialize other content types, you need to provide a custom deserializer using the *client_object.decoders* property. In the example below, `application/xml` responses are decoded using the `luarapidxml` library:

```
local http_client = require('http.client').new()
local xml = require("luarapidxml")

http_client.decoders = {
    ['application/xml'] = function(body, _content_type)
        return xml.decode(body)
    end,
}

local response = http_client:get('https://httpbin.org/xml')
local document = response:decode()
print("'title' value: "..document['attr']['title'])
```

The output for the code sample above should look as follows:

```
'title' value: Sample Slide Show
```

### Decompressing

The HTTP client can automatically decompress a response body based on the `Content-Encoding` header value. To enable this capability, pass the required formats using the *request_options.accept_encoding* option:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/gzip', {accept_encoding = "br,␣
↪gzip, deflate"})
print('Is response gzipped: '..tostring(response:decode()['gzipped']))
```

### Streaming download

The HTTP client supports chunked reading of request data. This can be achieved as follows:

1. Set the *chunked* option to `true`. In this case, a request method returns *io_object* instead of *response_object*.
2. Use the *io_object.read()* method to read data in chunks of a specified length or up to a specific delimiter.
3. Call the *io_object.finish()* method to finish reading data.

The example below shows how to get chunks of a JSON response sequentially instead of waiting for the entire response:

```
local http_client = require('http.client').new()
local json = require('json')

local io = http_client:get('https://httpbin.org/stream/5', {chunked = true})
local chunk_ids = ''
while data ~= '' do
    local data = io:read('\n')
    if data == '' then break end
    local decoded_data = json.decode(data)
    chunk_ids = chunk_ids..decoded_data['id']..' '
end
print('IDs of received chunks: '..chunk_ids)
io:finish()
```

### Redirects

By default, the HTTP client redirects to a URL provided in the `Location` header of a `3xx` response. If required, you can disable redirection using the *follow_location* option:

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/cookies/set?session_id=abc123&
↪csrftoken=u32t4o&', {follow_location = false})
```

### API Reference

| **Functions** | |
|---|---|
| *http.client.new()* | Создание экземпляра HTTP-клиента |
| **Objects** | |
| *client_options* | Configuration options of the client |
| *client_object* | An HTTP client instance |
| *request_options* | Options passed to a request |
| *response_object* | A response object |
| *io_object* | An IO object used to read/write data in chunks |

### http.client.new()

`http.client.new`([*options*])

Create an HTTP client instance.

**Параметры**

- `options` (*table*) – configuration options of the client (see *client_options*)

**возвращает** a new HTTP client instance (see *client_object*)

**тип возвращаемого значения** пользовательские данные

**Пример:**

```
local http_client = require('http.client').new()
```

### client_options

`object client_options`

Configuration options of the client. These options can be passed to the *http.client.new()* function.

`client_options.max_connections`

Specifies the maximum number of entries in the cache. This option affects libcurl CURLMOPT_MAXCONNECTS. The default is `-1`.

**Пример:**

```
local http_client = require('http.client').new({max_connections = 5})
```

**Note**

Do not set `max_connections` to less than `max_total_connections` unless you are confident about your actions. If `max_connections` is less than `max_total_connections`, libcurl doesn't reuse sockets in some cases for requests that go to the same host. If the limit is reached and a new request occurs, then libcurl creates a new socket first, sends the request, waits for the first connection to be free, and closes it to avoid exceeding the `max_connections` cache size. In the worst case, libcurl creates a new socket for every request, even if all requests go to the same host.

`client_options.max_total_connections`

Specifies the maximum number of active connections. This option affects libcurl CURLMOPT_MAX_TOTAL_CONNECTIONS.

**Note**

You may want to control the maximum number of sockets that a particular HTTP client uses simultaneously. If a system passes many requests to distinct hosts, then libcurl cannot reuse sockets. In this case, setting `max_total_connections` may be useful since it causes curl to avoid creating too many sockets, which would not be used anyway.

### client_object

`object client_object`

An HTTP client instance that exposes the API for *making requests*. To create the client, call *http.client.new()*.

`client_object:request(`*method*, *url*, *body*, *opts*`)`

Make an HTTP request and receive a response.

**Параметры**

- `method` (*string*) – a request HTTP method. Possible values: `GET`, `POST`, `PUT`, `PATCH`, `OPTIONS`, `HEAD`, `DELETE`, `TRACE`, `CONNECT`.
- `url` (*string*) – a request URL, for example, `https://httpbin.org/get`
- `body` (*string*) – a request body (see *Body*)
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method returns one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**Пример:**

```
local http_client = require('http.client').new()
local response = http_client:request('GET', 'https://httpbin.org/get')
```

**See also:** *Making requests*, *Receiving responses*

`client_object:get(`*url*, *opts*`)`
Make a `GET` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `https://httpbin.org/get`
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**Пример:**

```
local http_client = require('http.client').new()
local response = http_client:get('https://httpbin.org/get')
```

**See also:** *Making requests*, *Receiving responses*

`client_object:post(`*url*, *body*, *opts*`)`
Make a `POST` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `https://httpbin.org/post`
- `body` (*string*) – a request body (see *Body*)
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**Пример:**

```
local http_client = require('http.client').new()
local response = http_client:post('https://httpbin.org/anything', {
    user_id = 123,
    user_name = 'John Smith'
})
print('Posted data: '..response:decode()['data'])
```

**See also:** *Making requests*, *Receiving responses*

`client_object:put(`*url*, *body*, *opts*`)`
Make a `PUT` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `https://httpbin.org/put`
- `body` (*string*) – a request body (see *Body*)
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**See also:** *Making requests*, *Receiving responses*

`client_object:patch(`*url*, *body*, *opts*`)`
Make a `PATCH` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `https://httpbin.org/patch`
- `body` (*string*) – a request body (see *Body*)
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**See also:** *Making requests*, *Receiving responses*

`client_object:delete(`*url*, *opts*`)`
Make a `DELETE` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `https://httpbin.org/delete`
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**See also:** *Making requests*, *Receiving responses*

`client_object:head(`*url*, *opts*`)`
Make a `HEAD` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `https://httpbin.org/get`
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**See also:** *Making requests*, *Receiving responses*

`client_object:options(`*url*, *opts*`)`
Make an `OPTIONS` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `https://httpbin.org/get`
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**See also:** *Making requests*, *Receiving responses*

`client_object:trace(`*url*, *opts*`)`
Make a `TRACE` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `https://httpbin.org/get`
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**See also:** *Making requests*, *Receiving responses*

`client_object:connect(`*url*, *opts*`)`
Make a `CONNECT` request and receive a response.

**Параметры**

- `url` (*string*) – a request URL, for example, `server.example.com:80`
- `opts` (*table*) – request options (see *request_options*)

**возвращает** This method might return one of the following objects:

- *response_object*
- *io_object* if *request_options.chunked* is set to `true`

**тип возвращаемого значения** таблица

**See also:** *Making requests*, *Receiving responses*

`client_object:stat()`
Get a table with statistics for the HTTP client:

- `active_requests` – the number of currently executing requests
- `sockets_added` – the total number of sockets added into an event loop
- `sockets_deleted` – the total number of sockets deleted from an event loop

- `total_requests` – the total number of requests
- `http_200_responses` – the total number of requests that returned HTTP `200 OK` responses
- `http_other_responses` – the total number of requests that returned non-`200 OK` responses
- `failed_requests` – the total number of failed requests, including system, curl, and HTTP errors

`client_object.decoders`
**Since:** *2.11.0*

Decoders used to deserialize response data based on the `Content-Type` header value. Learn more from *Deserialization*.

### request_options

`object request_options`
Options passed to a request method (*request*, *get*, *post*, and so on).

**See also:** *Making requests*

`request_options.ca_file`
The path to an SSL certificate file to verify the peer with.

**тип возвращаемого значения** string

`request_options.ca_path`
The path to a directory holding one or more certificates to verify the peer with.

**тип возвращаемого значения** string

`request_options.chunked`
**Since:** *2.11.0*

Specifies whether an HTTP client should return the full response (*response_object*) or an IO object (*io_object*) used for streaming download/upload.

**тип возвращаемого значения** boolean

**See also:** *Streaming download*, *Streaming upload*

`request_options.headers`
A table of *HTTP headers* passed to a request.

**тип возвращаемого значения** таблица

`request_options.params`
**Since:** *2.11.0*

A table of parameters passed to a request. The behavior of this option depends on the request type, for example:

- For a *GET* request, this option specifies *query string parameters*.
- For a *POST* request, this option specifies *form parameters* to be sent using the `application/x-www-form-urlencoded` type.

**тип возвращаемого значения** таблица

`request_options.keepalive_idle`
A delay (in seconds) the operating system waits while the connection is idle before sending keepalive probes.

**тип возвращаемого значения** integer

**See also:** CURLOPT_TCP_KEEPIDLE, *keepalive_interval*

`request_options.keepalive_interval`
The interval (in seconds) the operating system waits between sending keepalive probes. If both *keepalive_idle* and `keepalive_interval` are set, then Tarantool also sets the HTTP keepalive headers: `Connection:Keep-Alive` and `Keep-Alive:timeout=<keepalive_idle>`. Otherwise, Tarantool sends `Connection:close`.

**тип возвращаемого значения** integer

**See also:** CURLOPT_TCP_KEEPINTVL

`request_options.low_speed_limit`
The average transfer speed in bytes per second that the transfer should be below during «low speed time» seconds for the library to consider it to be too slow and abort.

**тип возвращаемого значения** integer

**See also:** CURLOPT_LOW_SPEED_LIMIT

`request_options.low_speed_time`
The time that the transfer speed should be below the «low speed limit» for the library to consider it too slow and abort.

**тип возвращаемого значения** integer

**See also:** CURLOPT_LOW_SPEED_TIME

`request_options.max_header_name_len`
The maximum length of a header name. If a header name length exceeds this value, it is truncated to this length. The default value is `32`.

**тип возвращаемого значения** integer

`request_options.follow_location`
Specify whether the HTTP client follows redirect URLs provided in the `Location` header for `3xx` responses. When a non-`3xx` response is received, the client returns it as a result. If you set this option to `false`, the client returns the first `3xx` response.

**тип возвращаемого значения** boolean

**See also:** *Redirects*

`request_options.no_proxy`
A comma-separated list of hosts that do not require proxies, or `*`, or `''`.

- Set `no_proxy = host [, host ...]` to specify hosts that can be reached without requiring a proxy, even if `proxy` is set to a non-blank value and/or if a proxy-related environment variable has been set.
- Set `no__proxy = '*'` to specify that all hosts can be reached without requiring a proxy, which is equivalent to setting `proxy=''`.
- Set `no_proxy = ''` to specify that no hosts can be reached without requiring a proxy, even if a proxy-related environment variable (`HTTP_PROXY`) is used.

If `no_proxy` is not set, then a proxy-related environment variable (`HTTP_PROXY`) may be used.

**тип возвращаемого значения** string

**See also:** CURLOPT_NOPROXY

`request_options.proxy`
A proxy server host or IP address, or `''`.

- If `proxy` is a host or IP address, then it may begin with a scheme, for example, `https://` for an HTTPS proxy or `http://` for an HTTP proxy.
- If `proxy` is set to `''` an empty string, then proxy use is disabled, and no proxy-related environment variable is used.
- If `proxy` is not set, then a proxy-related environment variable may be used, such as `HTTP_PROXY` or `HTTPS_PROXY` or `FTP_PROXY`, or `ALL_PROXY` if the protocol can be any protocol.

**тип возвращаемого значения** string

**See also:** CURLOPT_PROXY

`request_options.proxy_port`
A proxy server port. The default is `443` for an HTTPS proxy and `1080` for a non-HTTPS proxy.

**тип возвращаемого значения** integer

**See also:** CURLOPT_PROXYPORT

`request_options.proxy_user_pwd`
A proxy server username and password. This option might have one of the following formats:

- `proxy_user_pwd = `*`user_name`*`:`
- `proxy_user_pwd = :`*`password`*
- `proxy_user_pwd = `*`user_name`*`:`*`password`*

**тип возвращаемого значения** string

**See also:** CURLOPT_USERPWD

`request_options.ssl_cert`
A path to an SSL client certificate file.

**тип возвращаемого значения** string

**See also:** CURLOPT_SSLCERT

`request_options.ssl_key`
A path to a private key file for a TLS and SSL client certificate.

**тип возвращаемого значения** string

**See also:** CURLOPT_SSLKEY

`request_options.timeout`
The number of seconds to wait for a curl API read request before timing out. The default timeout is set to infinity (`36586400100` seconds).

**тип возвращаемого значения** integer

`request_options.unix_socket`
A socket name to use instead of an Internet address for a local connection.

**тип возвращаемого значения** string

**Example:** `/tmp/unix_domain_socket.sock`

`request_options.verbose`
Turn on/off a verbose mode.

**тип возвращаемого значения** boolean

`request_options.verify_host`
Enable verification of the certificate's name (CN) against the specified host.

**тип возвращаемого значения** integer

**See also:** CURLOPT_SSL_VERIFYHOST

`request_options.verify_peer`
Set on/off verification of the peer's SSL certificate.

**тип возвращаемого значения** integer

**See also:** CURLOPT_SSL_VERIFYPEER

`request_options.accept_encoding`
Enable *decompression* of an HTTP response data based on the specified `Accept-Encoding` request header. You can pass the following values to this option:

- `''` – if an empty string is passed, the `Accept-Encoding` contains all the supported encodings (`identity`, `deflate`, `gzip`, and `br`).
- `br, gzip, deflate` – a comma-separated list of encodings passed in `Accept-Encoding`.

**тип возвращаемого значения** string

**See also:** CURLOPT_ACCEPT_ENCODING

### response_object

`object response_object`
A response object returned by a request method (*request*, *get*, *post*, and so on).

**See also:** *io_object*

`response_object.status`
A response status code.

**тип возвращаемого значения** integer

**See also:** *Status code*

`response_object.reason`
A response status text.

**тип возвращаемого значения** string

**See also:** *Status code*

`response_object.headers`
Response headers.

**тип возвращаемого значения** таблица

**See also:** *Headers*

`response_object.cookies`
Response cookies. The value is an array of two elements where the first one is the cookie value and the second one is an array with the cookie's options.

**тип возвращаемого значения** таблица

**See also:** *Cookies*

`response_object.body`
A response body. Use *decode* to decode the response body.

**тип возвращаемого значения** таблица

**See also:** *Response body*

`response_object.proto`
An HTTP protocol version.

**тип возвращаемого значения** string

`response_object:decode()`
**Since:** *2.11.0*

Decode the response body to a Lua object based on the content type.

**возвращает** a decoded body

**тип возвращаемого значения** таблица

**See also:** *Deserialization*

### io_object

`object io_object`
**Since:** *2.11.0*

An IO object used to read or write data in chunks. To get an IO object instead of the full response (*response_object*), you need to set the *chunked* request option to `true`.

`io_object:read(`*chunk*[, *timeout*]`)`
`io_object:read(`*delimiter*[, *timeout*]`)`
`io_object:read(`*{chunk = chunk, delimiter = delimiter}*[, *timeout*]`)`
Read request data in chunks of a specified length or up to a specific delimiter.

**Параметры**

- `chunk` (`integer`) – the maximum number of bytes to read
- `delimiter` (*`string`*) – the delimiter used to stop reading data
- `timeout` (`integer`) – the number of seconds to wait. The default is `10`.

**возвращает** A chunk of read data. Returns an empty string if there is nothing more to read.

**тип возвращаемого значения** *string*

**See also:** *Streaming download*

`io_object:write(`*data*[, *timeout*]`)`
Write the specified chunk of data.

**Параметры**

- `data` (*`table`*) – data to be written
- `timeout` (`integer`) – the number of seconds to wait. The default is `10`.

**See also:** *Streaming upload*

`io_object:finish(`[*timeout*]`)`
Finish reading or writing data.

**Параметры**

- `timeout` (`integer`) – the number of seconds to wait. The default is `10`.

**See also:** *Streaming download*, *Streaming upload*

### 9.5.18 Модуль iconv

#### Общие сведения

Модуль `iconv` предоставляет метод конвертации строки с одним типом кодировки в строку с другим типом кодировки, например из ASCII в UTF-8. Он основывается на процедурах с iconv в POSIX.

Точный список доступных кодировок зависит от окружения. Как правило, в список входят ASCII, BIG5, KOI8R, LATIN8, MS-GREEK, SJIS и около 100 других. Чтобы увидеть общий список, введите команду `iconv --list` в терминале.

#### Указатель

Ниже приведен перечень всех функций модуля `iconv`.

| Имя | Назначение |
|---|---|
| *iconv.new()* | Создание экземпляра iconv |
| *iconv.converter()* | Преобразование строки |

`iconv.new(`*to*, *from*`)`
Создание нового iconv-экземпляра.

**Параметры**

- `to` (*string*) – название будущей кодировки.
- `from` (*string*) – название используемой кодировки.

**возвращает** новый экземпляр iconv – на самом деле, вызываемая функция

**тип возвращаемого значения** пользовательские данные

Если значение одного из параметров представляет собой недопустимое имя, появится сообщение об ошибке.

**Пример:**

```
tarantool> converter = require('iconv').new('UTF8', 'ASCII')
---
...
```

`iconv.converter(`*input-string*`)`

Преобразование.

**param string input-string** строка для преобразования («из»)

**возвращает** строка, получаемая в результате преобразования («в»)

Если что-либо в строке input-string нельзя преобразовать, появится сообщение об ошибке, строка останется неизменной.

**Пример:**

We know that the Unicode code point for «Д» (CYRILLIC CAPITAL LETTER DE) is hexadecimal 0414 according to the character database of Unicode. Therefore that is what it will look like in UTF-16. We know that Tarantool typically uses the UTF-8 character set. So make a from-UTF-8-to-UTF-16 converter, use string.hex(„Д“) to show what Д's encoding looks like in the UTF-8 source, and use string.hex(„Д“-after-conversion) to show what it looks like in the UTF-16 target. Since the result is 0414, we see that iconv conversion works. (Different iconv implementations might use different names, for example UTF-16BE instead of UTF16BE.)

```
tarantool> string.hex('Д')
---
- d094
...

tarantool> converter = require('iconv').new('UTF16BE', 'UTF8')
---
...

tarantool> utf16_string = converter('Д')
---
...

tarantool> string.hex(utf16_string)
---
- '0414'
...
```

### 9.5.19 Module jit

#### Overview

The `jit` module has functions for tracing the LuaJIT Just-In-Time compiler's progress, showing the bytecode or assembler output that the compiler produces, and in general providing information about what LuaJIT does with Lua code.

#### Index

Below is a list of all `jit` functions.

---

**Примечание:**

**In this document, we will use:**

- `jit_dis_x64` for `require('jit.dis_x64')`,
- `jit_v` for `require('jit.v')`,
- `jit_dump` for `require('jit.dump')`.

---

| Name | Use |
|---|---|
| *jit_bc.dump()* | Print the byte code of a function |
| *jit_dis_x86.disass()* | Print the i386 assembler code of a string of bytes |
| *jit_dis_x64.disass()* | Print the x86-64 assembler code of a string of bytes |
| *jit_dump.on()*, *jit_dump.off()* | Print the intermediate or machine code of the following Lua code |
| *jit_v.on()*, *jit_v.off()* | Print a trace of LuaJIT's progress compiling and interpreting code |

`jit_bc.dump(`*function*`)`

Prints the byte code of a function.

**Example:**

```
tarantool> jit_bc = require('jit.bc')
---
...

tarantool> function f()
         > print("D")
         > end
---
...

tarantool> jit_bc.dump(f)
-- BYTECODE -- 0x01113163c8:1-3
0001    GGET     0   0      ; "print"
0002    KSTR     2   1      ; "D"
0003    CALL     0   1   2
0004    RET0     0   1

---
...
```

```
function f()
  print("D")
end
require('jit.bc').dump(f)
```

For a list of available options, read the source code of bc.lua.

`jit_dis_x86.disass(`*string*`)`

Prints the i386 assembler code of a string of bytes.

**Example:**

```
tarantool> -- Disassemble hexadecimal 97 which is the x86 code for xchg eax, edi
---
...

tarantool> jit_dis_x86 = require('jit.dis_x86')
---
...

tarantool> jit_dis_86.disass('\x97')
00000000  97                xchg eax, edi
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...
```

For a list of available options, read the source code of dis_x86.lua.

jit_dis_x64.disass(*string*)
: Prints the x86-64 assembler code of a string of bytes.

**Example:**

```
tarantool> -- Disassemble hexadecimal 97 which is the x86-64 code for xchg eax, edi
---
...

tarantool> jit_dis_x64 = require('jit.dis_x64')
---
...

tarantool> jit_dis_64.disass('\x97')
00000000  97                xchg eax, edi
---
...
```

For a list of available options, read the source code of dis_x64.lua.

jit_dump.on(*option*[, *output file*])
jit_dump.off()
: Prints the intermediate or machine code of the following Lua code.

**Example:**

```
tarantool> -- Show the machine code of a Lua "for" loop
tarantool> jit_dump = require('jit.dump')
tarantool> jit_dump.on('m')
tarantool> x = 0;
tarantool> for i = 1, 1e6 do
         > x = x + i
         > end
---- TRACE 1 start 0x01047fbc38:1
---- TRACE 1 mcode 148
104c29f6b  mov dword [r14-0xed0], 0x1
104c29f76  cvttsd2si ebp, [rdx]
104c29f7a  rorx rbx, [rdx-0x10], 0x2f
104c29f81  shr rbx, 0x11
104c29f85  mov rdx, [rbx+0x10]
104c29f89  cmp dword [rdx+0x34], +0x3f
104c29f8d  jnz 0x104c1a010  ->0
104c29f93  mov rcx, [rdx+0x28]
104c29f97  mov rdi, 0xfffd8001046b3d58
104c29fa1  cmp rdi, [rcx+0x320]
104c29fa8  jnz 0x104c1a010  ->0
104c29fae  lea rax, [rcx+0x318]
104c29fb5  cmp dword [rax+0x4], 0xfff90000
104c29fbc  jnb 0x104c1a010  ->0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
104c29fc2  xorps xmm7, xmm7
104c29fc5  cvtsi2sd xmm7, ebp
104c29fc9  addsd xmm7, [rax]
104c29fcd  movsd [rax], xmm7
104c29fd1  add ebp, +0x01
104c29fd4  cmp ebp, 0x000f4240
104c29fda  jg 0x104c1a014   ->1
->LOOP:
104c29fe0  xorps xmm6, xmm6
104c29fe3  cvtsi2sd xmm6, ebp
104c29fe7  addsd xmm7, xmm6
104c29feb  movsd [rax], xmm7
104c29fef  add ebp, +0x01
104c29ff2  cmp ebp, 0x000f4240
104c29ff8  jle 0x104c29fe0  ->LOOP
104c29ffa  jmp 0x104c1a01c  ->3
---- TRACE 1 stop -> loop

---
...

tarantool> print(x)
500000500000
---
...

tarantool> jit_dump.off()
---
...
```

For a list of available options, read the source code of dump.lua.

jit_v.on(*option*[, *output file*])
jit_v.off()

Prints a trace of LuaJIT's progress compiling and interpreting code.

**Example:**

```
tarantool> -- Show what LuaJIT is doing for a Lua "for" loop
tarantool> jit_v = require('jit.v')
tarantool> jit_v.on()
tarantool> l = 0
tarantool> for i = 1, 1e6 do
         >     l = l + i
         > end
[TRACE   3 "for i = 1, 1e6 do
    l = l + i
end":1 loop]
---
...

tarantool> print(l)
500000500000
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...

tarantool> jit_v.off()
---
...
```

For a list of available options, read the source code of v.lua.

### 9.5.20 Модуль json

#### Общие сведения

Модуль `json` определяет процедуры работы с форматом JSON. Он создан на основе модуля Lua-CJSON от Mark Pulford. Полное руководство по Lua-CJSON включено в официальную документацию.

#### Указатель

Ниже приведен перечень всех функций и элементов модуля `json`.

| Имя | Назначение |
|---|---|
| *json.encode()* | Конвертация Lua-объекта в JSON-строку |
| *json.decode()* | Конвертация JSON-строки в Lua-объект |
| *__serialize parameter* | Output structure specification |
| *json.cfg()* | Изменение конфигурации |
| *json.NULL* | Аналог «nil» в языке Lua |

`json.encode`(*lua-value*[, *configuration*])

Конвертация Lua-объекта в JSON-строку.

**Параметры**

- `lua_value` – скалярное значение или значение из Lua-таблицы.
- `configuration` – see *json.cfg*

**возвращает** оригинальное значение, преобразованное в JSON-строку.

**тип возвращаемого значения** строка

**Пример:**

```
tarantool> json=require('json')
---
...
tarantool> json.encode(123)
---
- '123'
...
tarantool> json.encode({123})
---
- '[123]'
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> json.encode({123, 234, 345})
---
- '[123,234,345]'
...
tarantool> json.encode({abc = 234, cde = 345})
---
- '{"cde":345,"abc":234}'
...
tarantool> json.encode({hello = {'world'}})
---
- '{"hello":["world"]}'
...
```

json.decode(*string*[, *configuration*])

Конвертация JSON-строки в Lua-объект.

**Параметры**

- string (*string*) – строка в формате JSON.
- configuration – see *json.cfg*

**возвращает** оригинальное содержание в формате Lua-таблицы.

**тип возвращаемого значения** таблица

**Пример:**

```
tarantool> json = require('json')
---
...
tarantool> json.decode('123')
---
- 123
...
tarantool> json.decode('[123, "hello"]')
---
- [123, 'hello']
...
tarantool> json.decode('{"hello": "world"}').hello
---
- world
...
```

Чтобы увидеть применение json.decode() в приложении, см. практическое задание *Подсчет суммы по JSON-полям во всех кортежах*.

**__serialize parameter:**

Структуру JSON-вывода можно указать с помощью __serialize:

- „seq“, „sequence“, „array“ - table encoded as an array
- „map“, „mapping“ - table encoded as a map
- function - the meta-method called to unpack serializable representation of table, cdata or userdata objects

Serializing „A“ and „B“ with different __serialize values brings different results:

```
tarantool> json.encode(setmetatable({'A', 'B'}, { __serialize="seq"}))
---
- '["A","B"]'
...
tarantool> json.encode(setmetatable({'A', 'B'}, { __serialize="map"}))
---
- '{"1":"A","2":"B"}'
...
tarantool> json.encode({setmetatable({f1 = 'A', f2 = 'B'}, { __serialize="map"})})
---
- '[{"f2":"B","f1":"A"}]'
...
tarantool> json.encode({setmetatable({f1 = 'A', f2 = 'B'}, { __serialize="seq"})})
---
- '[[]]'
...
```

json.cfg(*table*)
Set values that affect the behavior of *json.encode* and *json.decode*.

The values are all either integers or boolean `true`/`false`.

| Характеристика | Значение по умолчанию | Назначение |
|---|---|---|
| `cfg.encode_max_depth` | 128 | Max recursion depth for encoding |
| `cfg.encode_deep_as_nil` | false (strict) | A flag saying whether to crop tables with nesting level deeper than `cfg.encode_max_depth`. Not-encoded fields are replaced with one null. If not set, too deep nesting is considered an error. |
| `cfg.encode_invalid_numbers` | true | A flag saying whether to enable encoding of NaN and Inf numbers |
| `cfg.encode_number_precision` | 14 | Precision of floating point numbers |
| `cfg.encode_load_metatables` | true | A flag saying whether the serializer will follow *__serialize* metatable field |
| `cfg.encode_use_tostring` | false (strict) | A flag saying whether to use `tostring()` for unknown types |
| `cfg.encode_invalid_as_nil` | false (strict) | A flag saying whether use NULL for non-recognized types |
| `cfg.encode_sparse_convert` | true | A flag saying whether to handle excessively sparse arrays as maps. See detailed description *below*. |
| `cfg.encode_sparse_ratio` | 2 | 1/`encode_sparse_ratio` is the permissible percentage of missing values in a sparse array. |
| `cfg.encode_sparse_safe` | 10 | A limit ensuring that small Lua arrays are always encoded as sparse arrays (instead of generating an error or encoding as a map) |
| `cfg.decode_invalid_numbers` | true | A flag saying whether to enable decoding of NaN and Inf numbers |
| `cfg.decode_save_metatables` | true | A flag saying whether to set metatables for all arrays and maps |
| `cfg.decode_max_depth` | 128 | Max recursion depth for decoding |

**Sparse arrays features:**

During encoding, the JSON encoder tries to classify a table into one of four kinds:

- map - at least one table index is not unsigned integer
- regular array - all array indexes are available
- sparse array - at least one array index is missing
- excessively sparse array - the number of values missing exceeds the configured ratio

An array is excessively sparse when **all** the following conditions are met:

- `encode_sparse_ratio` > 0
- `max(table)` > `encode_sparse_safe`
- `max(table)` > `count(table)` * `encode_sparse_ratio`

The JSON encoder will never consider an array to be excessively sparse when `encode_sparse_ratio` = 0. The `encode_sparse_safe` limit ensures that small Lua arrays are always encoded as sparse arrays. By default, attempting to encode an excessively sparse array will generate an error. If `encode_sparse_convert` is set to `true`, excessively sparse arrays will be handled as maps.

**json.cfg() example 1:**

The following code will encode 0/0 as NaN («not a number») and 1/0 as Inf («infinity»), rather than returning nil or an error message:

```
json = require('json')
json.cfg{encode_invalid_numbers = true}
x = 0/0
y = 1/0
json.encode({1, x, y, 2})
```

Результат запроса `json.encode()` будет следующим:

```
tarantool> json.encode({1, x, y, 2})
---
- '[1,nan,inf,2]
...
```

**json.cfg example 2:**

To avoid generating errors on attempts to encode unknown data types as userdata/cdata, you can use this code:

```
tarantool> httpc = require('http.client').new()
---
...

tarantool> json.encode(httpc.curl)
---
- error: unsupported Lua type 'userdata'
...

tarantool> json.encode(httpc.curl, {encode_use_tostring=true})
---
- '"userdata: 0x010a4ef2a0"'
...
```

**Примечание:** To achieve the same effect for only one call to `json.encode()` (i.e. without changing the configuration permanently), you can use `json.encode({1, x, y, 2}, {encode_invalid_numbers = true})`.

Similar configuration settings exist for *MsgPack* and *YAML*.

`json.NULL`
Значение, сопоставимое с нулевым значением «nil» в языке Lua, которое можно использовать в качестве объекта-заполнителя в кортеже.

**Пример:**

```
-- Когда полю Lua-таблицы присваивается nil, это поле -- null
tarantool> {nil, 'a', 'b'}
---
- - null
  - a
  - b
...
-- Когда полю Lua-таблицы присваивается json.NULL, это поле --  json.NULL
tarantool> {json.NULL, 'a', 'b'}
---
- - null
  - a
  - b
...
-- Когда JSON-полю присваивается json.NULL, это поле -- null
tarantool> json.encode({field2 = json.NULL, field1 = 'a', field3 = 'c'})
---
- '{"field2":null,"field1":"a","field3":"c"}'
...
```

### 9.5.21 Модуль key_def

В модуле `key_def` есть функция, позволяющая создавать определение номеров и типов полей кортежа. Это определение обычно используется вместе с определением индекса, чтобы извлекать или сравнивать значения ключей индекса.

`key_def.new(`*parts*`)`
Создание нового экземпляра key_def.

**Параметры**

- `parts` (*table*) – типы и номера полей. В таблице `parts` должен быть хотя бы один элемент. Для каждого элемента обязательны только атрибуты `type` и `fieldno`/`field`.

**возвращает** *объект key_def*

The parts table has components which are the same as the `parts` option in *Options for space_object:create_index()*.

`fieldno` (целое число), например `fieldno = 1`. Вместо `fieldno` можно использовать `field`.

`type` (строка), например `type = 'string'`.

Остальные компоненты необязательны.

Пример №1: `key_def.new({{type = 'unsigned', fieldno = 1}})`

Пример №2: `key_def.new({{type = 'string', collation = 'unicode', field = 2}})`

`object key_def_object`

Объект key_def возвращается функцией *key_def.new()*. У него есть следующие методы: *extract_key()*, *compare()*, *compare_with_key()*, *merge()*, *totable()*.

`key_def_object:extract_key(`*tuple*`)`

Получение кортежа, содержащего только поля объекта `key_def`.

**Параметры**

- `tuple` (*table*) – кортеж или Lua-таблица с содержимым поля

**возвращает** поля, определенные для объекта `key_def`,

**Пример №1:**

```
--  Предположим, что у некоторого элемента пять полей:
-- 1, 99.5, 'X', nil, 99.5
-- Нас интересуют только два из них:
-- третье (строка) и первое (целое число).
-- Мы можем определить эти поля инструкцией k = key_def.new
--  и извлечь значения командой k:extract_key.

tarantool> key_def = require('key_def')
---
...

tarantool> k = key_def.new({{type = 'string', fieldno = 3},
>                           {type = 'unsigned', fieldno = 1}})
---
...

tarantool> k:extract_key({1, 99.5, 'X', nil, 99.5})
---
- ['X', 1]
...
```

**Пример №2**

```
-- Теперь предположим, что элемент является кортежем в спейсе.
-- У спейса есть составной индекс, построенный по полям 3 и 1.
-- Мы можем передать определение индекса в качестве аргумента функции key_def.
↪new,
-- а не заполнять определение, как в примере №1.
-- Результат будет тот же.
key_def = require('key_def')
box.schema.space.create('T')
i = box.space.T:create_index('I', {parts={3, 'string', 1, 'unsigned'}})
box.space.T:insert{1, 99.5, 'X', nil, 99.5}
k = key_def.new(i.parts)
k:extract_key(box.space.T:get({'X', 1}))
```

**Пример №3**

```
-- Проходим по всем кортежам во вторичном неуникальном индексе
-- и извлекаем из них значения по первичному ключу.
-- Затем удаляем значения, используя уникальный индекс.
-- Этот код должен входить в Lua-функцию.
local key_def_lib = require('key_def')
local s = box.schema.space.create('test')
local pk = s:create_index('pk')
local sk = s:create_index('test', {unique = false, parts = {
    {2, 'number', path = 'a'}, {2, 'number', path = 'b'}}})
s:insert{1, {a = 1, b = 1}}
s:insert{2, {a = 1, b = 2}}
local key_def = key_def_lib.new(pk.parts)
for _, tuple in sk:pairs({1}) do
    local key = key_def:extract_key(tuple)
    pk:delete(key)
end
```

key_def_object:compare(*tuple_1*, *tuple_2*)

Сравнение полей кортежей `tuple_1` и `tuple_2` по определённому ключу. Пользователю не нужно писать код для сравнения отдельных полей. Учитываются типы полей и параметры сортировки. Фактически сравниваются значения `extract_key(tuple_1)` и `extract_key(tuple_2)`.

**Параметры**

- `tuple1` (*table*) – кортеж или Lua-таблица с содержимым поля
- `tuple2` (*table*) – кортеж или Lua-таблица с содержимым поля

**возвращает** положительное число, если значения полей tuple_1 больше значений полей tuple_2 по ключу; 0, если они равны; отрицательное число, если значения полей tuple_1 меньше значений полей tuple_2 по ключу

**Пример:**

```
-- Результат этого кода будет 0
key_def = require('key_def')
k = key_def.new({{type = 'string', fieldno = 3, collation = 'unicode_ci'},
                 {type = 'unsigned', fieldno = 1}})
k:compare({1, 99.5, 'X', nil, 99.5}, {1, 99.5, 'x', nil, 99.5})
```

key_def_object:compare_with_key(*tuple_1*, *tuple_2*)

Сравнение полей кортежей `tuple_1` с полями кортежа `tuple_2` по заданному ключу. Аналогично *key_def_object:compare()*, за исключением того, что `tuple_2` содержит только поля ключа. Фактически это сравнение `extract_key(tuple_1)` с `tuple_2`.

**Параметры**

- `tuple1` (*table*) – кортеж или Lua-таблица с содержимым поля
- `tuple2` (*table*) – кортеж или Lua-таблица с содержимым поля

**возвращает** положительное число, если значения полей tuple_1 больше значений полей tuple_2 по ключу; 0, если они равны; отрицательное число, если значения полей tuple_1 меньше значений полей tuple_2 по ключу

**Пример:**

```
-- Результат этого кода будет 0
key_def = require('key_def')
k = key_def.new({{type = 'string', fieldno = 3, collation = 'unicode_ci'},
                 {type = 'unsigned', fieldno = 1}})
k:compare_with_key({1, 99.5, 'X', nil, 99.5}, {'x', 1})
```

key_def_object:merge(*other_key_def_object*)

Объединение основного объекта key_def_object с другим объектом other_key_def_object. Возвращает новый объект key_def_object, содержащий сначала все поля основного объекта key_def_object, а потом те поля объекта other_key_def_object, которых не было в основном объекте key_def_object.

**Параметры**

- other_key_def_object (*key_def_object*) – определение полей, которые нужно добавить

**возвращает** key_def_object

**Пример:**

```
-- Результатом этого кода будет определение ключа
-- по полям с fieldno = 3 и fieldno = 1
key_def = require('key_def')
k = key_def.new({{type = 'string', fieldno = 3}})
k2= key_def.new({{type = 'unsigned', fieldno = 1},
                 {type = 'string', fieldno = 3}})
k:merge(k2)
```

key_def_object:totable()

Возвращает таблицу с содержимым key_def_object. Функция противоположна функции key_def.new():

- key_def.new() принимает таблицу, а возвращает объект key_def.
- key_def_object:totable() принимает объект key_def, а возвращает таблицу.

Это удобно при подготовке входных данных для методов сериализации (_serialize).

**возвращает** таблица

**Пример:**

```
-- Результатом этого кода будет таблица с type = 'string', fieldno = 3
key_def = require('key_def')
k = key_def.new({{type = 'string', fieldno = 3}})
k:totable()
```

## 9.5.22 Модуль log

### Общие сведения

Tarantool provides a set of *options* used to configure logging in various ways: you can set a level of logging, specify where to send the log's output, configure a log format, and so on. The `log` module allows you to configure logging in your application and provides additional capabilities, for example, logging custom messages and rotating log files.

### Указатель

Ниже приведен перечень всех функций и элементов модуля `log`.

| Имя | Назначение |
|---|---|
| *log.cfg({})* | Configure a logger |
| *log.error()* *log.warn()* *log.info()* *log.verbose()* *log.debug()* | Log a message with the specified level |
| *log.pid()* | Получение PID регистратора записи в журнале |
| *log.rotate()* | Ротация файла журнала |
| *log.new()* | Create a new logger with the specified name |

`log.cfg`(*{}*)
Configure logging options. The following options are available:

- `level`: Specify the level of detail the log has.

  The example below shows how to set the log level to `verbose`:

  ```
  local log = require('log')
  log.cfg { level = 'verbose' }
  ```

  See also: *log.level*.

- `log`: Specify where to send the log's output, for example, to a file, pipe, or system logger.

  **Example 1: sending the log to the tarantool.log file**

  ```
  log.cfg { log = 'tarantool.log' }
  ```

  **Example 2: sending the log to a pipe**

  ```
  log.cfg { log = '| cronolog tarantool.log' }
  ```

  **Example 3: sending the log to syslog**

  ```
  log.cfg { log = 'syslog:server=unix:/dev/log' }
  ```

  See also: *log.to*.

- `nonblock`: If **true**, Tarantool does not block during logging when the system is not ready for writing, and drops the message instead.

  See also: *log.nonblock*.

- `format`: Specify the log format: „plain“ or „json“.

  See also: *log.format*.

- `modules`: Configure the specified log levels for different modules.

  See also: *log.modules*.

`log.error(`*message*`)`
`log.warn(`*message*`)`
`log.info(`*message*`)`
`log.verbose(`*message*`)`
`log.debug(`*message*`)`

Log a message with the specified logging level. You can learn more about the available levels from the *log.level* option description.

**Example**

The example below shows how to log a message with the `warn` level:

```
log.warn('Warning message')
```

**Параметры**

- `message` (`any`) – A log message.
  - A message can be a string.
  - A message may contain C-style format specifiers `%d` or `%s`. Example:

    ```
    log.info('Tarantool version: %s', box.info.version)
    ```

  - A message may be a scalar data type or a table. Example:

    ```
    log.error({ 500, 'Internal error' })
    ```

**возвращает** nil

Выходное значение будет представлять собой строку в журнале, которая содержит следующее:

- the current timestamp
- a module name
- „E“, „W“, „I“, „V“ or „D“ depending on the called function
- `message`

Note that the message will not be logged if the severity level corresponding to the called function is less than *log.level*.

`log.pid()`

**возвращает** A PID of a logger. You can use this PID to send a signal to a log rotation program, so it can rotate logs.

`log.rotate()`

Rotate the log. For example, you need to call this function to continue logging after a log rotation program renames or moves a file with the latest logs.

**возвращает** nil

`log.new(`*name*`)`

**Since:** *2.11.0*

Create a new logger with the specified name. You can configure a specific log level for a new logger using the *log.modules* configuration property.

**Параметры**

- `name` (*string*) – a logger name

**возвращает** a logger instance

**Example**

This example shows how to set the `verbose` level for `module1` and the `error` level for `module2` in a configuration file:

```
log:
  modules:
    module1: 'verbose'
    module2: 'error'
app:
  file: 'app.lua'
```

To create the `module1` and `module2` loggers in your application (`app.lua`), call the `new()` function:

```
-- Creates new loggers --
module1_log = require('log').new('module1')
module2_log = require('log').new('module2')
```

Then, you can call functions corresponding to different logging levels to make sure that events with severities above or equal to the given levels are shown:

```
-- Prints 'info' messages --
module1_log.info('Info message from module1')
--[[
[16300] main/103/interactive/module1 I> Info message from module1
---
...
--]]

-- Swallows 'debug' messages --
module1_log.debug('Debug message from module1')
--[[
---
...
--]]

-- Swallows 'info' messages --
module2_log.info('Info message from module2')
--[[
---
...
--]]
```

At the same time, the events with severities below the specified levels are swallowed.

Example on GitHub: log_new_modules.

### 9.5.23 Модуль merger

#### Общие сведения

The `merger` module takes a stream of tuples and provides access to them as tables.

#### Указатель

The four functions for creating a merger object instance are:

- *merger.new_tuple_source()*,
- *merger.new_buffer_source()*,
- *merger.new_table_source*,
- *merger.new(merger_source...)*.

The methods for using a merger object are:

- *merger_object:select()*,
- *merger_object:pairs()*.

`merger.new_tuple_source(`*gen*, *param*, *state*`)`
Create a new merger instance from a tuple source.

A tuple source just returns one tuple.

The generator function `gen()` allows creation of multiple tuples via an iterator.

The `gen()` function should return:

- state, tuple each time it is called and a new tuple is available,
- nil when no more tuples are available.

**Параметры**

- `gen` – function for iteratively returning tuples
- `param` – parameter for the gen function

**возвращает** merger-object *a merger object*

Example: see *merger_object:pairs()* method.

`merger.new_buffer_source(`*gen*, *param*, *state*`)`
Create a new merger instance from a buffer source.

Parameters and return: same as for *merger.new_tuple_source*.

To set up a buffer, or a series of buffers, use *the buffer module*.

`merger.new_table_source(`*gen*, *param*, *state*`)`
Create a new merger instance from a table source.

Parameters and return: same as for *merger.new_tuple_source*.

Example: see *merger_object:select()* method.

merger.new(*key_def*, *sources*, *options*)
Create a new merger instance from a merger source.

A merger source is created from a *key_def* object and a set of (tuple or buffer or table or merger) sources. It performs a kind of merge sort. It chooses a source with a minimal / maximal tuple on each step, consumes a tuple from this source, and repeats.

**Параметры**

- `key_def` – object created with `key_def`
- `source` – parameter for the `gen()` function
- `options` – `reverse=true` if descending, false or nil if ascending

**возвращает** merger-object *a merger object*

A `key_def` can be cached across requests with the same ordering rules (typically these would be requests accessing the same space).

Example: see *merger_object:pairs()* method.

object merger_object
A merger object is an object returned by:

- *merger.new_tuple_source()* or
- *merger.new_buffer_source()* or
- *merger.new_table_source* or
- *merger.new(merger_source...)*.

It has methods:

- *merger_object:select()* or
- *merger_object:pairs()*.

merger_object:select(\[*buffer*\[, *limit* \]\])
Access the contents of a merger object with familiar `select` syntax.

**Параметры**

- `buffer` – as in `net.box` client *conn:select* method
- `limit` – as in `net.box` client *conn:select* method

**возвращает** a table of tuples, similar to what `select` would return

**Example with new_table_source():**

```
-- Source via new_table_source, simple generator function
-- tarantool> s:select()
-- ---
-- - - [100]
--   - [200]
-- ...
merger=require('merger')
k=0
function merger_function(param)
  k = k + 1
  if param[k] == nil then return nil end
  return box.NULL, param[k]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  end
chunks={}
chunks[1] = {{100}} chunks[2] = {{200}} chunks[3] = nil
s = merger.new_table_source(merger_function, chunks)
s:select()
```

merger_object:pairs()

The `pairs()` method (or the equivalent `ipairs()` alias method) returns a luafun iterator. It is a Lua iterator, but also provides a set of handy methods to operate in functional style.

**Параметры**

- `tuple` (*table*) – tuple or Lua table with field contents

**возвращает** the tuples that can be found with a standard `pairs()` function

**Example with new_tuple_source():**

```
-- Source via new_tuple_source, from a space of tables
-- The result will look like this:
-- tarantool> so:pairs():totable()
-- ---
-- - - [100]
--   - [200]
-- ...
merger = require('merger')
box.schema.space.create('s')
box.space.s:create_index('i')
box.space.s:insert({100})
box.space.s:insert({200})
so = merger.new_tuple_source(box.space.s:pairs())
so:pairs():totable()
```

**Example with two mergers:**

```
-- Source via key_def, and table data

-- Create the key_def object
merger = require('merger')
key_def_lib = require('key_def')
key_def = key_def_lib.new({{
    fieldno = 1,
    type = 'string',
}})
-- Create the table source
data = {{'a'}, {'b'}, {'c'}}
source = merger.new_source_fromtable(data)
i1 = merger.new(key_def, {source}):pairs()
i2 = merger.new(key_def, {source}):pairs()
-- t1 will be 'a' (tuple 1 from merger 1)
t1 = i1:head():totable()
-- t3 will be 'c' (tuple 3 from merger 2)
t3 = i2:head():totable()
-- t2 will be 'b' (tuple 2 from merger 1)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
t2 = i1:head():totable()
-- i1:is_null() will be true (merger 1 ends)
i1:is_null()
-- i2:is_null() will be true (merger 2 ends)
i2:is_null()
```

**More examples:**

See https://github.com/Totktonada/tarantool-merger-examples which, in addition to discussing the merger API in detail, shows Lua code for handling many more situations than are in this manual's brief examples.

### 9.5.24 Модуль msgpack

#### Общие сведения

The `msgpack` module decodes *raw MsgPack strings* by converting them to Lua objects, and encodes Lua objects by converting them to raw MsgPack strings. Tarantool makes heavy internal use of MsgPack because tuples in Tarantool are *stored* as MsgPack arrays.

Besides, starting from version 2.10.0, the `msgpack` module enables creating a specific userdata Lua object – MsgPack object. The MsgPack object stores arbitrary MsgPack data, and can be created from *any Lua object* including another MsgPack object and from a *raw MsgPack string*. The MsgPack object has its own set of *methods* and *iterators*.

---

**Примечание:**

- *MsgPack* is short for MessagePack.
- A «raw MsgPack string» is a byte array formatted according to the MsgPack specification including type bytes and sizes. The type bytes and sizes can be made displayable with *string.hex()*, or the raw MsgPack strings can be converted to Lua objects by using the `msgpack` module methods.

---

### API Reference

Below is a list of `msgpack` members and related objects.

| **Members** | |
|---|---|
| *msgpack.encode(lua_value)* | Convert a Lua object to a raw MsgPack string |
| *msgpack.encode(lua_value,ibuf)* | Convert a Lua object to a raw MsgPack string in an ibuf |
| *msgpack.decode(msgpack_string)* | Convert a raw MsgPack string to a Lua object |
| *msgpack.decode(C_style_string_pointer)* | Convert a raw MsgPack string in an ibuf to a Lua object |
| *msgpack.decode_unchecked(msgpack_string)* | Convert a raw MsgPack string to a Lua object |
| *msgpack.decode_unchecked(C_style_string_pointer)* | Convert a raw MsgPack string to a Lua object |
| *msgpack.decode_array_header(byte-array, size)* | Call the MsgPuck's `mp_decode_array` function and return the array size and a pointer to the first array component |
| *msgpack.decode_map_header(byte-array, size)* | Call the MsgPuck's `mp_decode_map` function and return the map size and a pointer to the first map component |
| *__serialize* parameter | Output structure specification |
| *msgpack.cfg()* | Change MsgPack configuration settings |
| *msgpack.NULL* | Analog of Lua's `nil` |
| *msgpack.object(lua_value)* | Create a MsgPack object from a Lua object |
| *msgpack.object_from_raw(msgpack_string)* | Create a MsgPack object from a raw MsgPack string |
| *msgpack.object_from_raw(C_style_string_pointer, size)* | Create a MsgPack object from a raw MsgPack string |
| *msgpack.is_object(some_argument)* | Check if an argument is a MsgPack object |
| **Related objects** | |
| *msgpack_object* | A MsgPack object |
| *iterator_object* | A MsgPack iterator object |

### Members

`msgpack.encode(`*lua_value*`)`
Convert a Lua object to a raw MsgPack string.

**Параметры**

- `lua_value` – скалярное значение или значение из Lua-таблицы.

**возвращает** the original contents formatted as a raw MsgPack string;

**тип возвращаемого значения** raw MsgPack string

`msgpack.encode(`*lua_value*, *ibuf*`)`
Convert a Lua object to a raw MsgPack string in an ibuf, which is a buffer such as *buffer.ibuf()* creates. As with *encode(lua_value)*, the result is a raw MsgPack string, but it goes to the `ibuf` output instead of being returned.

**Параметры**

- `lua_value` (`lua-object`) – скалярное значение или значение из Lua-таблицы.
- `ibuf` (*buffer*) – (output parameter) where the result raw MsgPack string goes

**возвращает** number of bytes in the output

**тип возвращаемого значения** raw MsgPack string

Example using *buffer.ibuf()* and ffi.string() and *string.hex()*: The result will be „91a161“ because 91 is the MessagePack encoding of «fixarray size 1», a1 is the MessagePack encoding of «fixstr size 1», and 61 is the UTF-8 encoding of „a“:

```
ibuf = require('buffer').ibuf()
msgpack_string_size = require('msgpack').encode({'a'}, ibuf)
msgpack_string = require('ffi').string(ibuf.rpos, msgpack_string_size)
string.hex(msgpack_string)
```

msgpack.decode(*msgpack_string*[, *start_position*])

Convert a raw MsgPack string to a Lua object.

**Параметры**

- msgpack_string (*string*) – a raw MsgPack string.
- start_position (integer) – откуда начинать, минимальное значение = 1, максимальное = длина строки, по умолчанию = 1.

**возвращает**

- (if msgpack_string is a valid raw MsgPack string) the original contents of msgpack_string, formatted as a Lua object, usually a Lua table, (otherwise) a scalar value, such as a string or a number;
- «next_start_position». Если расшифровка decode останавливается после разбора байта N в msgpack_string, то «next_start_position» = N + 1, а decode(msgpack_string, next_start_position) продолжит разбор с места остановки предыдущего decode плюс 1. Как правило, decode разбирает всю строку msgpack_string, поэтому «next_start_position» будет равняться string.len(msgpack_string) + 1.

**тип возвращаемого значения** Lua object and number

Example: The result will be [„a“] and 4:

```
msgpack_string = require('msgpack').encode({'a'})
require('msgpack').decode(msgpack_string, 1)
```

msgpack.decode(*C_style_string_pointer*, *size*)

Convert a raw MsgPack string, whose address is supplied as a C-style string pointer such as the rpos pointer which is inside an ibuf such as *buffer.ibuf()* creates, to a Lua object. A C-style string pointer may be described as cdata<char *> or cdata<const char *>.

**Параметры**

- C_style_string_pointer (*buffer*) – a pointer to a raw MsgPack string.
- size (integer) – number of bytes in the raw MsgPack string

**возвращает**

- (if C_style_string_pointer points to a valid raw MsgPack string) the original contents of msgpack_string, formatted as a Lua object, usually a Lua table, (otherwise) a scalar value, such as a string or a number;
- returned_pointer = a C-style pointer to the byte after what was passed, so that C_style_string_pointer + size = returned_pointer

**тип возвращаемого значения** table and C-style pointer to after what was passed

Example using *buffer.ibuf* and pointer arithmetic: The result will be [„a“] and 3 and true:

```
ibuf = require('buffer').ibuf()
msgpack_string_size = require('msgpack').encode({'a'}, ibuf)
a, b = require('msgpack').decode(ibuf.rpos, msgpack_string_size)
a, b - ibuf.rpos, msgpack_string_size == b - ibuf.rpos
```

`msgpack.decode_unchecked(`*msgpack_string*[, *start_position*]`)`
Input and output are the same as for *decode(string)*.

`msgpack.decode_unchecked(`*C_style_string_pointer*`)`
Input and output are the same as for *decode(C_style_string_pointer)*, except that `size` is not needed. Some checking is skipped, and `decode_unchecked(C_style_string_pointer)` can operate with string pointers to buffers which `decode(C_style_string_pointer)` cannot handle. For an example see the *buffer* module.

`msgpack.decode_array_header(`*byte-array*, *size*`)`
Call the MsgPuck's `mp_decode_array` function and return the array size and a pointer to the first array component. A subsequent call to `msgpack_decode` can decode the component instead of the whole array.

**Параметры**

- `byte-array` – a pointer to a raw MsgPack string.
- `size` – a number greater than or equal to the string's length

**возвращает**

- the size of the array;
- a pointer to after the array header.

**Example:**

```
-- Example of decode_array_header
-- Suppose we have the raw data '\x93\x01\x02\x03'.
-- \x93 is MsgPack encoding for a header of a three-item array.
-- We want to skip it and decode the next three items.
msgpack = require('msgpack');
ffi = require('ffi');
x, y = msgpack.decode_array_header(ffi.cast('char*', '\x93\x01\x02\x03'), 4)
a = msgpack.decode(y, 1);
b = msgpack.decode(y + 1, 1);
c = msgpack.decode(y + 2, 1);
a, b, c
-- The result is: 1,2,3.
```

`msgpack.decode_map_header(`*byte-array*, *size*`)`
Call the MsgPuck's `mp_decode_map` function and return the map size and a pointer to the first map component. A subsequent call to `msgpack_decode` can decode the component instead of the whole map.

**Параметры**

- `byte-array` – a pointer to a raw MsgPack string.
- `size` – a number greater than or equal to the raw MsgPack string's length

**возвращает**

- the size of the map;
- a pointer to after the map header.

**Example:**

```
-- Example of decode_map_header
-- Suppose we have the raw data '\x81\xa2\x41\x41\xc3'.
-- '\x81' is MsgPack encoding for a header of a one-item map.
-- We want to skip it and decode the next map item.
msgpack = require('msgpack');
ffi = require('ffi')
x, y = msgpack.decode_map_header(ffi.cast('char*', '\x81\xa2\x41\x41\xc3'), 5)
a = msgpack.decode(y, 3);
b = msgpack.decode(y + 3, 1)
x, a, b
-- The result is: 1,"AA", true.
```

### __serialize parameter

The MsgPack output structure can be specified with the `__serialize` parameter:

- „seq“, „sequence“, „array“ – table encoded as an array
- „map“, „mappping“ – table encoded as a map
- function – the meta-method called to unpack the serializable representation of table, cdata, or userdata objects

Serializing „A“ and „B“ with different `__serialize` values brings different results. To show this, here is a routine which encodes `{'A','B'}` both as an array and as a map, then displays each result in hexadecimal.

```
function hexdump(bytes)
    local result = ''
    for i = 1, #bytes do
        result = result .. string.format("%x", string.byte(bytes, i)) .. ' '
    end
    return result
end

msgpack = require('msgpack')
m1 = msgpack.encode(setmetatable({'A', 'B'}, {
                              __serialize = "seq"
                            }))
m2 = msgpack.encode(setmetatable({'A', 'B'}, {
                              __serialize = "map"
                            }))
print('array encoding: ', hexdump(m1))
print('map encoding: ', hexdump(m2))
```

**Результат:**

```
**array** encoding: 92 a1 41 a1 42
**map** encoding:   82 01 a1 41 02 a1 42
```

The MsgPack Specification page explains that the first encoding means:

```
fixarray(2), fixstr(1), "A", fixstr(1), "B"
```

а значение второго результата кодирования:

```
fixmap(2), key(1), fixstr(1), "A", key(2), fixstr(2), "B"
```

Ниже приведены примеры всех стандартных типов: слева отображение в Lua-таблице, а справа – имя и кодировка в формате MsgPack.

**Стандартные типы в MsgPack-кодировке**

| | |
|---|---|
| {} | „fixmap“ = 80, если метатаблица – ассоциативный массив „map“, в противном случае, „fixarray“ = 90 |
| „a“ | „fixstr“ = a1 61 |
| false | „false“ = c2 |
| true | „true“ = c3 |
| 127 | „positive fixint“ = 7f |
| 65535 | „uint 16“ = cd ff ff |
| 4294967295 | „uint 32“ = ce ff ff ff ff |
| nil | „nil“ = c0 |
| msgpack.NULL | то же, что и nil |
| [0] = 5 | „fixmap(1)“ + „positive fixint“ (для ключа) + „positive fixint“ (для значения) = 81 00 05 |
| [0] = nil | „fixmap(0)“ = 80 – nil не хранится, если это отсутствующее значение ассоциативного массива |
| 1,5 | „float 64“ = cb 3f f8 00 00 00 00 00 00 |

`msgpack.cfg(`*table*`)`

Change MsgPack configuration settings.

The values are all either integers or boolean `true`/`false`.

| Характеристика | Значение по умолчанию | Назначение |
|---|---|---|
| `cfg.encode_max_depth` | 128 | The maximum recursion depth for encoding |
| `cfg.encode_deep_as_nil` | false | Specify whether to crop tables with nesting level deeper than `cfg.encode_max_depth`. Not-encoded fields are replaced with one null. If not set, too high nesting is considered an error. |
| `cfg.encode_invalid_numbers` | true | Specify whether to enable encoding of NaN and Inf numbers |
| `cfg.encode_load_metatables` | true | Specify whether the serializer will follow *__serialize* metatable field |
| `cfg.encode_use_tostring` | false | Specify whether to use `tostring()` for unknown types |
| `cfg.encode_invalid_as_nil` | false | Specify whether to use NULL for non-recognized types |
| `cfg.encode_sparse_convert` | true | Specify whether to handle excessively sparse arrays as maps. See detailed description *below* |
| `cfg.encode_sparse_ratio` | 2 | 1/`encode_sparse_ratio` is the permissible percentage of missing values in a sparse array |
| `cfg.encode_sparse_safe` | 10 | A limit ensuring that small Lua arrays are always encoded as sparse arrays (instead of generating an error or encoding as a map) |
| `cfg.encode_error_as_ext` | true | Specify how error objects (*box.error.new()*) are encoded in the MsgPack format:<br>• if `true`, errors are encoded as the the *MP_ERROR* MsgPack extension.<br>• if `false`, the encoding format depends on other configuration options (`encode_load_metatables`, `encode_use_tostring`, `encode_invalid_as_nil`). |
| `cfg.decode_invalid_numbers` | true | Specify whether to enable decoding of NaN and Inf numbers |
| `cfg.decode_save_metatables` | true | Specify whether to set metatables for all arrays and maps |

**Sparse arrays features**

During encoding, the MsgPack encoder tries to classify tables into one of four kinds:

- map - at least one table index is not unsigned integer
- regular array - all array indexes are available
- sparse array - at least one array index is missing
- excessively sparse array - the number of values missing exceeds the configured ratio

An array is excessively sparse when **all** the following conditions are met:

- `encode_sparse_ratio` > 0
- `max(table)` > `encode_sparse_safe`
- `max(table)` > `count(table)` * `encode_sparse_ratio`

MsgPack encoder never considers an array to be excessively sparse when `encode_sparse_ratio = 0`. The `encode_sparse_safe` limit ensures that small Lua arrays are always encoded as sparse arrays. By default, attempting to encode an excessively sparse array generates an error. If `encode_sparse_convert` is set to `true`, excessively sparse arrays will be handled as maps.

**msgpack.cfg() example 1:**

If `msgpack.cfg.encode_invalid_numbers = true` (the default), then NaN and Inf are legal values. If that is not desirable, then ensure that `msgpack.encode()` does not accept them, by saying `msgpack.cfg{encode_invalid_numbers = false}`, thus:

```
tarantool> msgpack = require('msgpack'); msgpack.cfg{encode_invalid_numbers = true}
---
...
tarantool> msgpack.decode(msgpack.encode{1, 0 / 0, 1 / 0, false})
---
- [1, -nan, inf, false]
- 22
...
tarantool> msgpack.cfg{encode_invalid_numbers = false}
---
...
tarantool> msgpack.decode(msgpack.encode{1, 0 / 0, 1 / 0, false})
---
- error: ... number must not be NaN or Inf'
...
```

**msgpack.cfg() example 2:**

To avoid generating errors on attempts to encode unknown data types as userdata/cdata, you can use this code:

```
tarantool> httpc = require('http.client').new()
---
...

tarantool> msgpack.encode(httpc.curl)
---
- error: unsupported Lua type 'userdata'
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> msgpack.cfg{encode_use_tostring = true}
---
...

tarantool> msgpack.encode(httpc.curl)
---
- !!binary tnVzZXJkYXRhOiAweDAxMDU5NDQ2Mzg=
...
```

**Примечание:** To achieve the same effect for only one call to `msgpack.encode()` (that is without changing the configuration permanently), you can use `msgpack.new({encode_invalid_numbers = true}).encode({1, 2})`.

Similar configuration settings exist for *JSON* and *YAML*.

`msgpack.NULL`
: Значение, сопоставимое с нулевым значением «nil» в языке Lua, которое можно использовать в качестве объекта-заполнителя в кортеже.

  **Пример:**

```
tarantool> msgpack = require('msgpack')
---
...
tarantool> y = msgpack.encode({'a',1,'b',2})
---
...
tarantool> z = msgpack.decode(y)
---
...
tarantool> z[1], z[2], z[3], z[4]
---
- a
- 1
- b
- 2
...
tarantool> box.space.tester:insert{20, msgpack.NULL, 20}
---
- [20, null, 20]
...
```

`msgpack.object(`*lua_value*`)`
: **Since:** *2.10.0*

  Encode an arbitrary Lua object into the MsgPack format.

  **Параметры**

  - `lua_value` (`lua-object`) – a Lua object of any type.

  **возвращает** encoded MsgPack data encapsulated in a MsgPack object.

  **тип возвращаемого значения** userdata

**Example:**

```
local msgpack = require('msgpack')

-- Create a MsgPack object from a Lua object of any type
local mp_from_number = msgpack.object(123)
local mp_from_string = msgpack.object('hello world')
local mp_from_array = msgpack.object({ 10, 20, 30 })
local mp_from_table = msgpack.object({ band_name = 'The Beatles', year = 1960 })
local mp_from_tuple = msgpack.object(box.tuple.new{1, 'The Beatles', 1960})
```

msgpack.object_from_raw(*msgpack_string*)

**Since:** *2.10.0*

Create a MsgPack object from a raw MsgPack string.

**Параметры**

- msgpack_string (*string*) – a raw MsgPack string.

**возвращает** a MsgPack object

**тип возвращаемого значения** userdata

**Example:**

```
local msgpack = require('msgpack')

-- Create a MsgPack object from a raw MsgPack string
local raw_mp_string = msgpack.encode({ 10, 20, 30 })
local mp_from_mp_string = msgpack.object_from_raw(raw_mp_string)
```

msgpack.object_from_raw(*C_style_string_pointer*, *size*)

**Since:** *2.10.0*

Create a MsgPack object from a raw MsgPack string. The address of the MsgPack string is supplied as a C-style string pointer such as the `rpos` pointer inside an `ibuf` that the *buffer.ibuf()* creates. A C-style string pointer may be described as `cdata<char *>` or `cdata<const char *>`.

**Параметры**

- C_style_string_pointer (*buffer*) – a pointer to a raw MsgPack string.
- size (integer) – number of bytes in the raw MsgPack string.

**возвращает** a MsgPack object

**тип возвращаемого значения** userdata

**Example:**

```
local msgpack = require('msgpack')

-- Create a MsgPack object from a raw MsgPack string using buffer
local buffer = require('buffer')
local ibuf = buffer.ibuf()
msgpack.encode({ 10, 20, 30 }, ibuf)
local mp_from_mp_string_pt = msgpack.object_from_raw(ibuf.buf, ibuf:size())
```

msgpack.is_object(*some_argument*)

**Since:** *2.10.0*

Check if the given argument is a MsgPack object.

**Параметры**

- `some_agrument` – any argument.

**возвращает** `true` if the argument is a MsgPack object; otherwise, `false`

**тип возвращаемого значения** boolean

**Example:**

```
local msgpack = require('msgpack')

local mp_from_string = msgpack.object('hello world')

-- Check if the given argument is a MsgPack object
local mp_is_object = msgpack.is_object(mp_from_string) -- Returns true
local string_is_object = msgpack.is_object('hello world') -- Returns false
```

### Related objects

#### msgpack_object

`object msgpack_object`

A MsgPack object that stores arbitrary MsgPack data. To create a MsgPack object from a Lua object or string, use the following methods:

- *msgpack.object*
- *msgpack.object_from_raw*

If a MsgPack object stores an array, it can be inserted into a database space:

```
box.space.bands:insert(msgpack.object({1, 'The Beatles', 1960}))
```

`msgpack_object:decode()`

**Since:** *2.10.0*

Decode MsgPack data in the MsgPack object.

**возвращает** a Lua object

**тип возвращаемого значения** Lua object

**Пример:**

```
local msgpack = require('msgpack')

local mp_from_number = msgpack.object(123)
local mp_from_string = msgpack.object('hello world')

-- Decode MsgPack data
local mp_number_decoded = mp_from_number:decode() -- Returns 123
local mp_string_decoded = mp_from_string:decode() -- Returns 'hello world'
```

`msgpack_object:iterator()`

**Since:** *2.10.0*

Create an iterator over the MsgPack data.

**возвращает** an *iterator object* over the MsgPack data

**тип возвращаемого значения** userdata

`msgpack_object[key]`
**Since:** *2.11.0*

Get an element of the MsgPack array by the specified index key. You can also use the *get(key)* method to get an array element.

The index key used to get the array element might be one of the following:

- if a MsgPack object is an array, the `key` is an integer value (starting with 1) that specifies the element index.
- if a MsgPack object is an associative array, `key` is the string value that specifies the element key. In this case, you can also access the array element using dot notation (`msgpack_object.<key>`).

If the specified key is missing in the array, `msgpack_object[key]` returns `nil`.

**Пример:**

```
local msgpack = require('msgpack')

local mp_from_array = msgpack.object({ 10, 20, 30 })
local mp_from_table = msgpack.object({ band_name = 'The Beatles', year = 1960 })
local mp_from_tuple = msgpack.object(box.tuple.new{1, 'The Beatles', 1960})

-- Get MsgPack data by the specified index or key
local mp_array_get_by_index = mp_from_array[1] -- Returns 10
local mp_table_get_by_key = mp_from_table['band_name'] -- Returns 'The Beatles'
local mp_table_get_by_nonexistent_key = mp_from_table['rating'] -- Returns nil
local mp_tuple_get_by_index = mp_from_tuple[3] -- Returns 1960
```

---

**Примечание:** Note that if the key for an associative array coincides with any `msgpack_object`'s method name, for example, „iterator“, `mp_from_table['iterator']` returns the `iterator` method function instead of a value corresponding to the „iterator“ key.

---

`msgpack_object:get(`*key*`)`
**Since:** *2.11.0*

Get an element of the MsgPack array by the specified index key. You can also use the indexed notation (*msgpack_ object[key]*) to get an array element.

**Параметры**

- `key` (`number/string`) – the index key used to get the array element, which might be one of the following:
  - if a MsgPack object is an array, the `key` is an integer value (starting with 1) that specifies the element index.
  - if a MsgPack object is an associative array, `key` is the string value that specifies the element key.

**возвращает** an element of the MsgPack array. If the specified key is missing in the array, `get` returns `nil`.

### iterator_object

object iterator_object
An iterator over a MsgPack array.

iterator_object:decode_array_header()
**Since:** *2.10.0*

Decode a MsgPack array header under the iterator cursor and advance the cursor. After calling this function, the iterator points to the first element of the array or to the value following the array if the array is empty.

**возвращает** number of elements in the array

**тип возвращаемого значения** number

**Possible errors:** raise an error if the type of the value under the iterator cursor is not `MP_ARRAY`.

**Пример:**

```
local msgpack = require('msgpack')

local mp_array = msgpack.object({ 10, 20, 30, 40 })
local mp_array_iterator = mp_array:iterator()

local size = mp_array_iterator:decode_array_header()  -- returns 4
local first = mp_array_iterator:decode()              -- returns 10
local second = mp_array_iterator:decode()             -- returns 20
mp_array_iterator:skip()                              -- returns none, skips 30
local fourth = mp_array_iterator:decode()             -- returns 40
```

iterator_object:decode_map_header()
**Since:** *2.10.0*

Decode a MsgPack map header under the iterator cursor and advance the cursor. After calling this function, the iterator points to the first key stored in the map or to the value following the map if the map is empty.

**возвращает** number of key-value pairs in the map

**тип возвращаемого значения** number

**Possible errors:** raise an error if the type of the value under the iterator cursor is not `MP_MAP`.

**Пример:**

```
local msgpack = require('msgpack')

local mp_map = msgpack.object({ foo = 123 })
local mp_map_iterator = mp_map:iterator()

local size = mp_map_iterator:decode_map_header() -- returns 1
local first = mp_map_iterator:decode()           -- returns 'foo'
local second = mp_map_iterator:decode()          -- returns '123'
```

iterator_object:decode()
**Since:** *2.10.0*

Decode a MsgPack value under the iterator cursor and advance the cursor.

**возвращает** a Lua object corresponding to the MsgPack value

**тип возвращаемого значения** Lua object

**Possible errors:** raise a Lua error if there's no data to decode.

**Пример:**

```
local msgpack = require('msgpack')

local mp_array = msgpack.object({ 10, 20, 30, 40 })
local mp_array_iterator = mp_array:iterator()

local size = mp_array_iterator:decode_array_header()  -- returns 4
local first = mp_array_iterator:decode()              -- returns 10
local second = mp_array_iterator:decode()             -- returns 20
mp_array_iterator:skip()                              -- returns none, skips 30
local fourth = mp_array_iterator:decode()             -- returns 40
```

`iterator_object:take()`

**Since:** *2.10.0*

Return a MsgPack value under the iterator cursor as a MsgPack object without decoding and advance the cursor. The method doesn't copy MsgPack data. Instead, it takes a reference to the original object.

**Possible errors:** raise a Lua error if there's no data to decode.

**Пример:**

```
local msgpack = require('msgpack')

local mp_array = msgpack.object({ 10, 20, 30 })
local mp_array_iterator = mp_array:iterator()

local size = mp_array_iterator:decode_array_header()  -- returns 3
local first = mp_array_iterator:decode()              -- returns 10
mp_array_iterator:skip()                              -- returns none, skips 20
local mp_value_under_cursor = mp_array_iterator:take()
local third = mp_value_under_cursor:decode()          -- returns 30
```

`iterator_object:take_array(`*count*`)`

**Since:** *2.10.0*

Copy the specified number of MsgPack values starting from the iterator's cursor position to a new MsgPack array object and advance the cursor.

**Параметры**

- `count` (`number`) – the number of MsgPack values to copy

**возвращает** a new MsgPack object

**Possible errors:** raise a Lua error if there aren't enough values to decode. In this case, the iterator's cursor position doesn't change.

**Пример:**

```
local msgpack = require('msgpack')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
local mp_array = msgpack.object({ 10, 20, 30, 40 })
local mp_array_iterator = mp_array:iterator()

local size = mp_array_iterator:decode_array_header()  -- returns 4
local first = mp_array_iterator:decode()              -- returns 10
local mp_array_new = mp_array_iterator:take_array(2)
local mp_array_new_decoded = mp_array_new:decode()    -- returns {20, 30}
local fourth = mp_array_iterator:decode()             -- returns 40
```

`iterator_object:skip()`

**Since:** *2.10.0*

Advance the iterator cursor by skipping one MsgPack value under the cursor. Returns nothing.

**Possible errors:** raise a Lua error if there's no data to skip.

**Пример:**

```
local msgpack = require('msgpack')

local mp_array = msgpack.object({ 10, 20, 30, 40 })
local mp_array_iterator = mp_array:iterator()

local size = mp_array_iterator:decode_array_header()  -- returns 4
local first = mp_array_iterator:decode()              -- returns 10
local second = mp_array_iterator:decode()             -- returns 20
mp_array_iterator:skip()                              -- returns none, skips 30
local fourth = mp_array_iterator:decode()             -- returns 40
```

### 9.5.25 Модуль net.box

#### Общие сведения

Модуль `net.box` включает в себя коннекторы для удаленных систем с базами данных. Одним из вариантов, который рассматривается позднее, является подключение к MySQL, MariaDB или PostgreSQL (см. справочник по *Модулям СУБД SQL*). Другим вариантом, который рассматривается в данном разделе, является подключение к экземплярам Tarantool-сервера по сети. Для быстрого знакомства с `net.box` вы можете обратиться к разделу *Начало работы с net.box*.

Можно вызвать следующие методы:

- `require('net.box')` – для получения объекта `net.box` (который называется `net_box` для примеров в данном разделе);
- `net_box.connect()` – для подключения и получения объекта подключения (который называется `conn` для примеров в данном разделе);
- другие процедуры `net.box()`, передающие `conn:` для выполнения запросов в удаленной системе базы данных;
- `conn:close` – для отключения.

Все методы `net.box` безопасны для файберов, то есть можно безопасно обмениваться и использовать один и тот же объект подключения в нескольких файберах одновременно. Фактически так лучше всего работать в Tarantool. Когда несколько файберов используют одно соединение, все запросы передаются по одному сетевому сокету, но каждый файбер получает правильный ответ. Уменьшение количества

активных сокетов снижает затрату ресурсов на системные вызовы и увеличивает общую производительность сервера. Однако, в некоторых случаях отдельного соединения недостаточно — например, когда необходимо отдавать приоритет разным запросам или использовать различные идентификаторы при аутентификации.

В большинстве методов `net.box` можно использовать последний аргумент `{options}`, который может быть следующим:

- `{timeout=...}`. Например, метод с последним аргументом `{timeout=1.5}` остановится через 1,5 секунды на локальном узле, хотя это не гарантирует, что выполнение остановится на удаленном сервере.
- `{buffer=...}`. Например, см. *модуль buffer*.
- `{is_async=...}`. Например, метод с заключительным аргументом `{is_async=true}` не будет ждать результата выполнения запроса. См. описание *is_async*.
- `{on_push=... on_push_ctx=...}`. Для получения внеполосных сообщений. См. описание *box.session.push()*.
- `{return_raw=...}` (since version 2.10.0). If set to `true`, net.box returns response data wrapped in a *MsgPack object* instead of decoding it to Lua. The default value is `false`. For an example, see option description *below*.

На диаграмме ниже представлены возможные состояния и варианты перехода из одного состояния в другое:

На этой диаграмме:

- Метод `net_box.connect()` запускает рабочий файбер, который устанавливает соединение и запускает рабочий процесс.
- The state machine goes to the `initial` state.
- Authentication and schema upload. It is possible later on to re-enter the `fetch_schema` state from `active` to trigger schema reload.
- The state changes to the `graceful_shutdown` state when the state machine receives a *box.shutdown* event from the remote host (see *conn:on_shutdown()*). Once all pending requests are completed, the state machine switches to the `error` (`error_reconnect`) state.
- The transport goes to the `error` state in case of an error. It can happen, for example, if the server closed the connection. If the `reconnect_after` option is set, instead of the 'error' state, the transport goes to the `error_reconnect` state.
- `conn.close()` method sets the state to `closed` and kills the worker. If the transport is already in the `error` state, `close()` does nothing.

### Указатель

Ниже приведен перечень всех функций модуля `net.box`.

| Имя | Назначение |
|---|---|
| *net_box.connect() net_box.new() net_box.self* | Создание подключения |
| *conn:ping()* | Выполнение команды проверки состояния PING |
| *conn:wait_connected()* | Ожидание активности или закрытия подключения |
| *conn:is_connected()* | Проверка активности или закрытия подключения |
| *conn:wait_state()* | Ожидание нужного состояния |
| *conn:close()* | Закрытие подключения |
| *conn.space.space-name:select{field-value}* | Выбор одного или нескольких кортежей |
| *conn.space.space-name:get{field-value}* | Выбор кортежа |
| *conn.space.space-name:insert{field-value}* | Вставка кортежа |
| *conn.space.space-name:replace{field-value}* | Вставка или замена кортежа |
| *conn.space.space-name:update{field-value}* | Обновление кортежа |
| *conn.space.space-name:upsert{field-value}* | Обновление кортежа |
| *conn.space.space-name:delete{field-value}* | Удаление кортежа |
| *conn:eval()* | Оценка и выполнение выражения в строке |
| *conn:call()* | Вызов хранимой процедуры |
| *conn:watch()* | Subscribe to events broadcast by a remote host |
| *conn:on_connect()* | Определение триггера на подключение |
| *conn:on_disconnect()* | Определение триггера на отключение |
| *conn:on_shutdown()* | Define a shutdown trigger |
| *conn:on_schema_reload()* | Определение триггера при изменении схемы |
| *conn:new_stream()* | Create a stream |
| *stream:begin()* | Begin a stream transaction |
| *stream:commit()* | Commit a stream transaction |
| *stream:rollback()* | Rollback a stream transaction |

net_box.connect(*URI*[, *{option[s]}*])

Создание нового подключения. Подключение устанавливается по требованию во время первого запроса. Можно повторно установить подключение автоматически после отключения (см. ниже опцию `reconnect_after`). Возвращается объект `conn`, который поддерживает методы создания удаленных запросов, таких как select, update или delete.

**Параметры**

- `URI` (*string*) – *URI* объекта подключения
- `options` – the supported options are shown below:
  - `user/password`: two options to connect to a remote host other than through *URI*. For example, instead of `connect('username:userpassword@localhost:3301')` you can write `connect('localhost:3301', {user = 'username', password='userpassword'})`.
  - `wait_connected`: a connection timeout. By default, the connection is blocked until the connection is established, but if you specify `wait_connected=false`, the connection returns immediately. If you specify this timeout, it will wait before returning (`wait_connected=1.5` makes it wait at most 1.5 seconds).

---

**Примечание:** Если присутствует `reconnect_after`, `wait_connected` проигнорирует неустойчивые отказы. Ожидание заканчивается, когда подключение установлено или явным образом закрыто.

---

- `reconnect_after`: a number of seconds to wait before reconnecting. The default value, as with the other `connect` options, is `nil`. If `reconnect_after` is greater than zero, then a `net.box` instance will attempt to reconnect if a connection is lost or a connection attempt fails. This makes transient network failures transparent to the application. Reconnection happens automatically in the background, so requests that initially fail due to connection drops fail, are transparently retried. The number of retries is unlimited, connection retries are made after any specified interval (for example, `reconnect_after=5` means that reconnect attempts are made every 5 seconds). When a connection is explicitly closed or when the Lua garbage collector removes it, then reconnect attempts stop.
- `call_16`: [since 1.7.2] a new binary protocol command for CALL in `net.box` connections by default. The new CALL is not backward compatible with previous versions. It no longer restricts a function to returning an array of tuples and allows returning an arbitrary MsgPack/JSON result, including scalars, nil and void (nothing). The old CALL is left intact for backward compatibility. It will not be present in the next major release. All programming language drivers will gradually be switched to the new CALL. To connect to a Tarantool instance that uses the old CALL, specify `call_16=true`.
- `connect_timeout`: a number of seconds to wait before returning «error: Connection timed out».
- `fetch_schema`: a boolean option that controls fetching schema changes from the server. Default: `true`. If you don't operate with remote spaces, for example, run only `call` or `eval`, set `fetch_schema` to `false` to avoid fetching schema changes which is not needed in this case.

  **Важно:** In connections with `fetch_schema == false`, remote spaces are unavailable and the *on_schema_reload* triggers don't work.

- `required_protocol_version`: a minimum version of the *IPROTO protocol* supported by the server. If the version of the *IPROTO protocol* supported by the server is lower than specified, the connection will fail with an error message. With `required_protocol_version = 1`, all connections fail where the *IPROTO protocol* version is lower than 1.
- `required_protocol_features`: specified *IPROTO protocol features* supported by the server. You can specify one or more `net.box` features from the table below. If the server does not support the specified features, the connection will fail with an error message. With `required_protocol_features = {'transactions'}`, all connections fail where the server has `transactions: false`.

| net.box feature | Назначение | IPROTO feature ID | IPROTO versions supporting the feature |
|---|---|---|---|
| `streams` | Requires streams support on the server | IPROTO_FEATURE_STREAMS | 3 and newer |
| `transactions` | Requires transactions support on the server | IPROTO_FEATURE_TRANSACTIONS | 3 and newer |
| `error_extension` | Requires support for *MP_ERROR* MsgPack extension on the server | IPROTO_FEATURE_ERROR_EXTENSION | 2 and newer |
| `watchers` | Requires remote *watchers* support on the server | IPROTO_FEATURE_WATCHERS | 3 and newer |

To learn more about IPROTO features, see *IPROTO_ID* and the *IPROTO_FEATURES* key.

**возвращает** объект подключения

**тип возвращаемого значения** пользовательские данные

**Примеры:**

```
net_box = require('net.box')

conn = net_box.connect('localhost:3301')
conn = net_box.connect('127.0.0.1:3302', {wait_connected = false})
conn = net_box.connect('127.0.0.1:3303', {reconnect_after = 5, call_16 = true})
conn = net_box.connect('127.0.0.1:3304', {required_protocol_version = 4, required_
↪protocol_features = {'transactions', 'streams'}, })
```

net_box.new(*URI*[, *{option[s]}*])

new() is a synonym for connect(). It is retained for backward compatibility. For more information, see the description of *net_box.connect()*.

object self

Для локального Tarantool-сервера есть заданный объект всегда установленного подключения под названием *net_box*.self. Он создан с целью облегчить полиморфное использование API модуля net_box. Таким образом, conn = *net_box*.connect('localhost:3301') можно заменить на conn = *net_box*.self.

Однако, есть важно отличие встроенного подключения от удаленного:

- With the embedded connection, requests which do not modify data do not yield. When using a remote connection, due to *the implicit rules* any request can yield, and the database state may have changed by the time it regains control.
- Не учитывается ни один параметр, передаваемый в запросе (is_async, on_push, timeout).

object conn

conn:ping([*options*])

Выполнение команды проверки состояния PING.

**Параметры**

- options (*table*) – поддерживается опция timeout=*секунды*

**возвращает** true (правда), если выполнено, false (ложь) в случае ошибки

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
net_box.self:ping({timeout = 0.5})
```

conn:wait_connected([*timeout*])

Ожидание активности или закрытия подключения.

**Параметры**

- timeout (number) – в секундах

**возвращает** true (правда) при подключении, false (ложь), если не выполнено.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
net_box.self:wait_connected()
```

`conn:is_connected()`
Проверка активности или закрытия подключения.

**возвращает** true (правда) при подключении, false (ложь), если не выполнено.

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
net_box.self:is_connected()
```

`conn:wait_state(`*state[s]*`[, `*timeout*`])`
[с 1.7.2] Ожидание нужного состояния.

**Параметры**

- `states` (*string*) – необходимое состояние
- `timeout` (`number`) – в секундах

**возвращает** true (правда) при подключении, false (ложь) при окончании времени ожидания или закрытии подключения

**тип возвращаемого значения** boolean (логический)

**Примеры:**

```
-- бесконечное ожидание состояния 'active':
conn:wait_state('active')

-- ожидание в течение максимум 1,5 секунд:
conn:wait_state('active', 1.5)

-- бесконечное ожидание состояния `active` или `fetch_schema`:
conn:wait_state({active=true, fetch_schema=true})
```

`conn:close()`
Закрытие подключения.

Объекты подключения удаляются сборщиком мусора в Lua, как и любой другой Lua-объект, поэтому удалять их явным образом необязательно. Однако, поскольку close() представляет собой системный вызов, лучше всего закрыть соединение явным образом, когда оно больше не используется, с целью ускорения работы сборщика мусора.

**Пример:**

```
conn:close()
```

`conn.space.<space-name>:select({field-value, ...} [, {options}])`
`conn.space.`*имя-спейса*`:select({...})` – это удаленный вызов, аналогичный локальному вызову `box.space.`*имя-спейса*`:select{...}` (*детали*). В дополнение см. *Модуль buffer и skip-header*.

**Пример:**

```
conn.space.testspace:select({1,'B'}, {timeout=1})
```

**Примечание:** Due to *the implicit yield rules* a local `box.space.space-name:select{...}` does not yield, but a remote `conn.space.space-name:select{...}` call does yield, so global variables or database tuples data may change when a remote `conn.space.space-name:select{...}` occurs.

`conn.space.<space-name>:get({field-value, ...} [, {options}])`
`conn.space.имя-спейса:get(...)` – это удаленный вызов, аналогичный локальному вызову `box.space.имя-спейса:get(...)` (*детали*).

**Пример:**

```
conn.space.testspace:get({1})
```

`conn.space.<space-name>:insert({field-value, ...} [, {options}])`
`conn.space.имя-спейса:insert(...)` – это удаленный вызов, аналогичный локальному вызову `box.space.имя-спейса:insert(...)` (*детали*). В дополнение см. *Модуль buffer и skip-header*.

**Пример:**

```
conn.space.testspace:insert({2,3,4,5}, {timeout=1.1})
```

`conn.space.<space-name>:replace({field-value, ...} [, {options}])`
`conn.space.имя-спейса:replace(...)` – это удаленный вызов, аналогичный локальному вызову `box.space.имя-спейса:replace(...)` (*детали*). В дополнение см. *Модуль buffer и skip-header*.

**Пример:**

```
conn.space.testspace:replace({5,6,7,8})
```

`conn.space.<space-name>:update({field-value, ...} [, {options}])`
`conn.space.имя-спейса:update(...)` – это удаленный вызов, аналогичный локальному вызову `box.space.имя-спейса:update(...)`.

**Пример:**

```
conn.space.Q:update({1},{{'=',2,5}}, {timeout=0})
```

`conn.space.<space-name>:upsert({field-value, ...} [, {options}])`
`conn.space.space-name:upsert(...)` is the remote-call equivalent of the local call `box.space.space-name:upsert(...)`. (*see details*). For an additional option see *Module buffer and skip-header*.

`conn.space.<space-name>:delete({field-value, ...} [, {options}])`
`conn.space.имя-спейса:delete(...)` – это удаленный вызов, аналогичный локальному вызову `box.space.имя-спейса:delete(...)`.(*детали*). В дополнение см. *Модуль buffer и skip-header*.

`conn:eval(`*Lua-string*`[, {`*arguments*`}[, {`*options*`}]])`
`conn:eval(Lua-строка)` оценивает и выполняет выражение в Lua-строке, которое может представлять собой любое выражение или несколько выражений. Требуются *права на выполнение*; если у пользователя таких прав нет, администратор может их выдать с помощью `box.schema.user.grant(имя-пользователя, 'execute', 'universe')`.

Чтобы гарантировать, что `conn:eval` вернет то, что возвращает выражение на Lua, начните Lua-строку со слова «return» (вернуть).

**Примеры:**

```
tarantool> --Lua-строка
tarantool> conn:eval('function f5() return 5+5 end; return f5();')
---
- 10
...
tarantool> --Lua-строка, {аргументы}
tarantool> conn:eval('return ...', {1,2,{3,'x'}})
---
- 1
- 2
- [3, 'x']
...
tarantool> --Lua-строка, {аргументы}, {парметры}
tarantool> conn:eval('return {nil,5}', {}, {timeout=0.1})
---
- [null, 5]
...
```

conn:call(*function-name*[, *{arguments}*[, *{options}*]])

conn:call('func', {'1', '2', '3'}) – это удаленный вызов, аналогичный func('1', '2', '3'). Таким образом, conn:call представляет собой удаленный вызов хранимой процедуры. conn:call возвращает то, что возвращает функция.

Ограничение: вызванная функция не может вернуть функцию, например, если func2 определяется как function func2 () return func end, то conn:call(func2) вернет ошибку «error: unsupported Lua type „function“».

**Примеры:**

```
tarantool> -- создание 2 функций с conn:eval()
tarantool> conn:eval('function f1() return 5+5 end;')
tarantool> conn:eval('function f2(x,y) return x,y end;')
tarantool> -- вызов первой функции без параметров и опций
tarantool> conn:call('f1')
---
- 10
...
tarantool> -- вызов второй функции с двумя параметрами и одной опцией
tarantool> conn:call('f2',{1,'B'},{timeout=99})
---
- 1
- B
...
```

conn:watch(*key*, *func*)

Subscribe to events broadcast by a remote host.

**Параметры**

- key (*string*) – a key name of an event to subscribe to
- func (function) – a callback to invoke when the key value is updated

**возвращает** a watcher handle. The handle consists of one method – unregister(), which unregisters the watcher.

To read more about watchers, see the *Functions for watchers* section.

The method has the same syntax as the *box.watch()* function, which is used for subscribing to events locally.

Watchers survive reconnection (see the `reconnect_after` connection *option*). All registered watchers are automatically resubscribed when the connection is reestablished.

If a remote host supports watchers, the `watchers` key will be set in the connection `peer_protocol_features`. For details, check the *net.box features table*.

---

**Примечание:** Keep in mind that garbage collection of a watcher handle doesn't lead to the watcher's destruction. In this case, the watcher remains registered. It is okay to discard the result of `watch` function if the watcher will never be unregistered.

---

**Example 1:**

Server:

```
-- Broadcast value 42 for the 'foo' key.
box.broadcast('foo', 42)
```

Client:

```
conn = net.box.connect(URI)
local log = require('log')
-- Subscribe to updates of the 'foo' key.
w = conn:watch('foo', function(key, value)
    assert(key == 'foo')
    log.info("The box.id value is '%d'", value)
end)
```

If you don't need the watcher anymore, you can unregister it using the command below:

```
w:unregister()
```

**Example 2:**

The `net.box module` provides the ability to monitor *updates of a configuration* stored in a Tarantool-based configuration storage by watching path or prefix changes. In the example below, *conn:watch()* is used to monitor updates of a configuration stored by the `/myapp/config/all` path:

```
net_box = require('net.box')
local conn = net_box.connect('127.0.0.1:4401')
local log = require('log')
conn:watch('config.storage:/myapp/config/all', function(key, value)
    log.info("Configuration stored by the '/myapp/config/all' key is changed")
end)
```

You can find the full example here: config_storage.

`conn:request(`*... {is_async=...}*`)`

`{is_async=true|false}` – это опция, которую можно применить во всех запросах `net_box`, включая `conn:call`, `conn:eval` и запросы `conn.space.space-name`.

По умолчанию, `is_async=false`, что означает, что запросы будут синхронными для файбера. Файбер блокируется в ожидании ответа на запрос или до истечения времени ожидания. До версии Tarantool 1.10 единственным способом выполнения асинхронных запросов было использование отдельных файберов.

`is_async=true` означает, что запросы будут асинхронными для файбера. Запрос вызывает передачу управления, но файбер не входит в режим ожидания. Сразу же возвращается результат, но это будет не результат запроса, а объект, который может использовать вызывающая программа для получения результат запроса.

У такого сразу же возвращаемого объекта, который мы называем «future» (будущий), есть собственные методы:

- `future:is_ready()` вернет true (правда), если доступен результат запроса,
- `future:result()` используется для получения результата запроса (возвращает ответ на запрос или **nil** в случае, если ответ еще не готов или произошла какая-либо ошибка),
- `future:wait_result(timeout)` будет ждать, когда результат запроса будет доступен, а затем получит его или выдаст ошибку, если по истечении времени ожидания результат не получен.
- `future:discard()` откажется от объекта.

В обычной ситуации пользователь введет команду `future=имя-запроса(... {is_async=true})`, а затем либо цикл с проверкой `future:is_ready()` до тех пор, пока он не вернет true, и получением результата с помощью `request_result=future:result()`, либо же команду `request_result=future:wait_result(...)`. Возможен вариант, когда клиент проверяет наличие внеполосных сообщений от сервера, вызывая в цикле `pairs()` – см. *box.session.push()*.

Можно использовать `future:discard()`, чтобы соединение забыло об ответе – если получен ответ для отброшенного объекта, то он будет проигнорирован, так что размер таблицы запросов будет уменьшен, а другие запросы будут выполняться быстрее.

**Примеры:**

```
-- Insert a tuple asynchronously --
tarantool> future = conn.space.bands:insert({10, 'Queen', 1970}, {is_async=true}
↪)
---
...
tarantool> future:is_ready()
---
- true
...
tarantool> future:result()
---
- [10, 'Queen', 1970]
...

-- Iterate through a space with 10 records to get data in chunks of 3 records --
tarantool> while true do
               future = conn.space.bands:select({}, {limit=3, after=position,␣
↪fetch_pos=true, is_async=true})
               result = future:wait_result()
               tuples = result[1]
               position = result[2]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                if position == nil then
                    break
                end
                print('Chunk size: '..#tuples)
            end
Chunk size: 3
Chunk size: 3
Chunk size: 3
Chunk size: 1
---
...
```

Как правило, `{is_async=true}` используется только при большой загрузке (более 100 000 запросов в секунду) и большой задержке чтения (более 1 секунды), или же при необходимости отправки нескольких одновременных запросов, которые собирают ответы (что иногда называется «отображение-свертка»).

---

**Примечание:** Хотя окончательный результат асинхронного запроса не отличается от результата синхронного запроса, у него другая структура: таблица, а не неупакованные значения.

---

`conn:request(`*... {return_raw=...}*`)`

`{return_raw=true}` is ignored for:

- Methods that return `nil`: `begin`, `commit`, `rollback`, `upsert`, `prepare`.
- `index.count` (returns number).

For `execute`, the option is applied only to data (`rows`). Metadata is decoded even if `{return_raw=true}`.

**Пример:**

```
local c = require('net.box').connect(uri)
local mp = c.eval('eval ...', {1, 2, 3}, {return_raw = true})
mp:decode() -- {1, 2, 3}
```

The option can be useful if you want to pass a response through without decoding or with partial decoding. The usage of *MsgPack object* can reduce pressure on the Lua garbage collector.

`conn:new_stream(`[*options*]`)`

Create a stream.

**Пример:**

```
-- Start a server to create a new stream
local conn = net_box.connect('localhost:3301')
local conn_space = conn.space.test
local stream = conn:new_stream()
local stream_space = stream.space.test
```

`object stream`

`stream:begin([txn_isolation])`
Begin a stream transaction. Instead of the direct method, you can also use the `call`, `eval` or execute methods with SQL transaction.

**Параметры**

- `txn_isolation` – *transaction isolation level*

`stream:commit()`
Commit a stream transaction. Instead of the direct method, you can also use the `call`, `eval` or execute methods with SQL transaction.

**Примеры:**

```
-- Begin stream transaction
stream:begin()
-- In the previously created ``accounts`` space with the primary key ``test``,␣
↪modify the fields 2 and 3
stream.space.accounts:update(test_1, {{'-', 2, 370}, {'+', 3, 100}})
-- Commit stream transaction
stream:commit()
```

`stream:rollback()`
Rollback a stream transaction. Instead of the direct method, you can also use the `call`, `eval` or execute methods with SQL transaction.

**Пример:**

```
-- Test rollback for memtx space
space:replace({1})
-- Select return tuple that was previously inserted, because this select␣
↪belongs to stream transaction
space:select({})
stream:rollback()
-- Select is empty, stream transaction rollback
space:select({})
```

### Триггеры

В модуле `net.box` можно использовать следующие *триггеры*:

`conn:on_connect([trigger-function[, old-trigger-function]])`
Define a trigger for execution when a new connection is established, and authentication and schema fetch are completed due to an event such as `net_box.connect`.

If a trigger function issues `net_box` requests, they must be *asynchronous* (`{is_async = true}`). An attempt to wait for request completion with `future:pairs()` or `future:wait_result()` in the trigger function will result in an error.

If the trigger execution fails and an exception happens, the connection's state changes to „error“. In this case, the connection is terminated, regardless of the `reconnect_after` option's value. Can be called as many times as reconnection happens, if `reconnect_after` is greater than zero.

**Параметры**

- `trigger-function` (`function`) – the trigger function. Takes the `conn` object as the first argument.

- `old-trigger-function` (`function`) – an existing trigger function to replace with `trigger-function`

**возвращает** nil или указатель функции

`conn:on_disconnect([`*trigger-function*`[,` *old-trigger-function*`]])`

Определение триггера, исполняемого после закрытия соединения. Если функция с триггером вызывает ошибку, то ошибка записывается в журнал, в противном случае записей не будет. Выполнение прекращается после явного закрытия соединения или удаления сборщиком мусора в Lua.

**Параметры**

- `trigger-function` (`function`) – the trigger function. Takes the `conn` object as the first argument
- `old-trigger-function` (`function`) – an existing trigger function to replace with `trigger-function`

**возвращает** nil или указатель функции

`conn:on_shutdown([`*trigger-function*`[,` *old-trigger-function*`]])`

Define a trigger for shutdown when a *box.shutdown* event is received.

The trigger starts in a new fiber. While the `on_shutdown()` trigger is running, the connection stays active. It means that the trigger callback is allowed to send new requests.

After the trigger return, the `net.box` connection goes to the `graceful_shutdown` state (check *the state diagram* for details). In this state, no new requests are allowed. The connection waits for all pending requests to be completed.

Once all in-progress requests have been processed, the connection is closed. The state changes to `error` or `error_reconnect` (if the `reconnect_after` option is defined).

Servers that do not support the `box.shutdown` event or *IPROTO_WATCH* just close the connection abruptly. In this case, the `on_shutdown()` trigger is not executed.

**Параметры**

- `trigger-function` (`function`) – the trigger function. Takes the `conn` object as the first argument
- `old-trigger-function` (`function`) – an existing trigger function to replace with `trigger-function`

**возвращает** nil или указатель функции

`conn:on_schema_reload([`*trigger-function*`[,` *old-trigger-function*`]])`

Определение триггера, исполняемого во время выполнения определенной операции на удаленном сервере после обновления схемы. Другими словами, если запрос к серверу не выполняется из-за ошибки несовпадения версии схемы, происходит перезагрузка схемы.

If a trigger function issues `net_box` requests, they must be *asynchronous* (`{is_async = true}`). An attempt to wait for request completion with `future:pairs()` or `future:wait_result()` in the trigger function will result in an error.

**Параметры**

- `trigger-function` (`function`) – the trigger function. Takes the `conn` object as the first argument
- `old-trigger-function` (`function`) – an existing trigger function to replace with `trigger-function`

**возвращает** nil или указатель функции

---

**Примечание:** Если указаны параметры `(nil, old-trigger-function)`, старый триггер будет удален.

Если не указан ни один параметр, ответом будет список существующих функций с триггером.

Find the detailed information about triggers in the *triggers* section.

---

### 9.5.26 Модуль os

#### Общие сведения

Модуль `os` включает в себя следующие функции: *execute()*, *rename()*, *getenv()*, *remove()*, *date()*, *exit()*, *time()*, *clock()*, *tmpname()*, *environ()*, *setenv()*, *setlocale()*, *difftime()*. Большинство этих функций описаны в Главе 22 руководства по языку Lua Библиотека функций операционной системы.

#### Указатель

Ниже приведен перечень всех функций модуля `os`.

| Имя | Назначение |
|---|---|
| *os.execute()* | Выполнение путем передачи в ОС |
| *os.rename()* | Переименование файла или директории |
| *os.getenv()* | Получение переменной окружения |
| *os.remove()* | Удаление файла или директории |
| *os.date()* | Получение даты в формате |
| *os.exit()* | Выход из программы |
| *os.time()* | Получение числа секунд с начала отсчета |
| *os.clock()* | Получение числа времени ЦП в секундах с момента начала программы |
| *os.tmpname()* | Получение имени временного файла |
| *os.environ()* | Получение таблицы со всеми переменными окружения |
| *os.setenv()* | Определение переменной окружения |
| *os.setlocale()* | Изменение локали |
| *os.difftime()* | Получение числа секунд между двумя значениями времени |

`os.execute(`*shell-command*`)`

Выполнение путем передачи в ОС.

**Параметры**

- `shell-command` (*string*) – что выполнить.

**Пример:**

```
tarantool> os.execute('ls -l /usr')
total 200
drwxr-xr-x   2 root root 65536 Apr 22 15:49 bin
drwxr-xr-x  59 root root 20480 Apr 18 07:58 include
drwxr-xr-x 210 root root 65536 Apr 18 07:59 lib
drwxr-xr-x  12 root root  4096 Apr 22 15:49 local
drwxr-xr-x   2 root root 12288 Jan 31 09:50 sbin
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...
```

os.rename(*old-name*, *new-name*)
Переименование файла или директории.

**Параметры**

- old-name (*string*) – имя существующего файла или директории,
- new-name (*string*) – измененное имя файла или директории.

**Пример:**

```
tarantool> os.rename('local','foreign')
---
- null
- 'local: No such file or directory'
- 2
...
```

os.getenv(*variable-name*)
Получение переменной окружения.

Параметры: (string) variable-name = имя переменной окружения.

**Пример:**

```
tarantool> os.getenv('PATH')
---
- /usr/local/sbin:/usr/local/bin:/usr/sbin
...
```

os.remove(*name*)
Удаление файла или директории.

Parameters: (string) name = имя файла или директории, которые будут удалены.

**Пример:**

```
tarantool> os.remove('file')
---
- true
...
```

os.date(*format-string*[, *time-since-epoch*])
Возврат даты в формате.

Parameters: (string) format-string = инструкции; (string) time-since-epoch = число секунд с 1970-01-01. Если не указать time-since-epoch, предполагается использование текущего времени.

**Пример:**

```
tarantool> os.date("%A %B %d")
---
- Sunday April 24
...
```

os.exit()
Выход из программы. Если выполняется на экземпляре сервера, останавливается работа экземпляра.

**Пример:**

```
tarantool> os.exit()
user@user-shell:~/tarantool_sandbox$
```

os.time()
Возврат числа секунд с начала отсчета.

**Пример:**

```
tarantool> os.time()
---
- 1461516945
...
```

os.clock()
Возврат числа времени ЦП в секундах с момента начала программы.

**Пример:**

```
tarantool> os.clock()
---
- 0.05
...
```

os.tmpname()
Возврат имени временного файла.

**Пример:**

```
tarantool> os.tmpname()
---
- /tmp/lua_7SW1m2
...
```

os.environ()
Возврат таблицы со всеми переменными окружения.

**Пример:**

```
tarantool> os.environ()['TERM']..os.environ()['SHELL']
---
- xterm/bin/bash
...
```

os.setenv(*variable-name*, *variable-value*)
Определение переменной окружения.

**Пример:**

```
tarantool> os.setenv('VERSION','99')
---
-
...
```

`os.setlocale(`[*new-locale-string*]`)`
Изменение локали. Если не указать new-locale-string, вернется текущая локаль.

**Пример:**

```
tarantool> string.sub(os.setlocale(),1,20)
---
- LC_CTYPE=en_US.UTF-8
...
```

`os.difftime(`*time1*, *time2*`)`
Возврат числа секунд между двумя значениями времени.

**Пример:**

```
tarantool> os.difftime(os.time() - 0)
---
- 1486594859
...
```

### 9.5.27 Модуль pickle

#### Указатель

Ниже приведен перечень всех функций модуля `pickle`.

| Имя | Назначение |
|---|---|
| *pickle.pack()* | Конвертация Lua-переменных в двоичный формат |
| *pickle.unpack()* | Конвертация Lua-переменных в двоичный формат |

`pickle.pack(`*format*, *argument*[, *argument ...*]`)`
To use Tarantool binary protocol primitives from Lua, it's necessary to convert Lua variables to binary format. The `pickle.pack()` helper function is prototyped after Perl pack.

**Спецификаторы формата**

| | |
|---|---|
| b, B | конвертирует скалярное Lua-значение в 1-байтное целое число и хранит целое число в полученной строке |
| s, S | конвертирует скалярное Lua-значение в 2-байтное целое число и хранит целое число в полученной строке, сначала младший байт |
| i, I | конвертирует скалярное Lua-значение в 4-байтное целое число и хранит целое число в полученной строке, сначала младший байт |
| l, L | конвертирует скалярное Lua-значение в 8-байтное целое число и хранит целое число в полученной строке, сначала младший байт |
| n | конвертирует скалярное Lua-значение в 2-байтное целое число и хранит целое число в полученной строке, порядок от старшего к младшему, |
| N | конвертирует скалярное Lua-значение в 4-байтное целое число и хранит целое число в полученной строке, порядок от старшего к младшему, |
| q, Q | конвертирует скалярное Lua-значение в 8-байтное целое число и хранит целое число в полученной строке, порядок от старшего к младшему, |
| f | конвертирует скалярное Lua-значение в 4-байтное число с плавающей запятой и хранит число с плавающей запятой в полученной строке |
| d | конвертирует скалярное Lua-значение в 8-байтное число двойной точности и хранит число двойной точности в полученной строке |
| a, A | конвертирует скалярное Lua-значение в последовательность байтов и хранит последовательность в полученной строке |

**Параметры**

- `format` (*string*) – строка со спецификаторами формата
- `argument(s)` (`scalar-value`) – скалярные значения к форматированию

**возвращает** бинарная строка, которая содержит все аргументы, упакованные в соответствии со спецификаторами формата.

**тип возвращаемого значения** строка

Скалярное значение может быть либо переменной, либо литеральным значением. Следует помнить, что большие целые числа нужно вводить с *tonumber64() или суффиксами LL или ULL*.

Возможные ошибки: неизвестный спецификатор формата.

**Пример:**

```
tarantool> pickle = require('pickle')
---
...
tarantool> box.space.tester:insert{0, 'hello world'}
---
- [0, 'hello world']
...
tarantool> box.space.tester:update({0}, {{'=', 2, 'bye world'}})
---
- [0, 'bye world']
...
tarantool> box.space.tester:update({0}, {
         >   {'=', 2, pickle.pack('iiA', 0, 3, 'hello')}
         > })
---
- [0, "\0\0\0\0\x03\0\0\0hello"]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> box.space.tester:update({0}, {{'=', 2, 4}})
---
- [0, 4]
...
tarantool> box.space.tester:update({0}, {{'+', 2, 4}})
---
- [0, 8]
...
tarantool> box.space.tester:update({0}, {{'^', 2, 4}})
---
- [0, 12]
...
```

pickle.unpack(*format*, *binary-string*)

Противоположность pickle.pack(). Внимание: если используется спецификатор формата „A“, он должен идти последним.

**Параметры**

- format (*string*) –
- binary-string (*string*) –

**возвращает** Список строк или чисел.

**тип возвращаемого значения** таблица

**Пример:**

```
tarantool> pickle = require('pickle')
---
...
tarantool> tuple = box.space.tester:replace{0}
---
...
tarantool> string.len(tuple[1])
---
- 1
...
tarantool> pickle.unpack('b', tuple[1])
---
- 48
...
tarantool> pickle.unpack('bsi', pickle.pack('bsi', 255, 65535, 4294967295))
---
- 255
- 65535
- 4294967295
...
tarantool> pickle.unpack('ls', pickle.pack('ls', tonumber64('18446744073709551615'),
↪ 65535))
---
...
tarantool> num, num64, str = pickle.unpack('slA', pickle.pack('slA', 666,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        > tonumber64('666666666666666'), 'string'))
---
...
```

### 9.5.28 Модуль popen

#### Общие сведения

Начиная с версии *2.4.1* в Tarantool есть встроенный модуль `popen`, предназначенный для выполнения внешних программ. Он работает аналогично модулю subprocess() в Python или Open3 в Ruby. Однако в `popen` нет вспомогательных средств, которые предоставляют эти языки; он предоставляет только базовые функции. Для создания объекта `popen` использует системный вызов vfork(), поэтому вызывающий поток блокируется до тех пор, пока не начинается выполнение дочернего процесса.

В модуле `popen` есть две функции для создания объекта popen:

- *popen.shell*, аналогичная системному вызову popen из libc;
- *popen.new* для создания объекта popen с более специфическими параметрами.

Обе функции возвращают дескриптор, который мы будем называть `popen_handle` или `ph`. Через дескриптор вы можете выполнять методы.

#### Указатель

Ниже приведен перечень всех функций `popen` и методов дескриптора `ph`.

| Имя | Назначение |
|---|---|
| *popen.shell()* | Выполнение shell-команды |
| *popen.new()* | Выполнение дочерней программы в новом процессе |
| *popen_handle:read()* | Считывание данных из дочернего процесса |
| *popen_handle:write()* | Запись строки в поток stdin дочернего процесса |
| *popen_handle:shutdown()* | Закрытие канала с std* со стороны родителя |
| *popen_handle:terminate()* | Отправка сигнала SIGTERM дочернему процессу |
| *popen_handle:kill()* | Отправка сигнала SIGKILL дочернему процессу |
| *popen_handle:signal()* | Отправка сигнала дочернему процессу |
| *popen_handle:info()* | Получение информации о дескрипторе popen |
| *popen_handle:wait()* | Ожидание, пока дочерний процесс не завершится или не получит сигнал |
| *popen_handle:close()* | Закрытие дескриптора popen |
| *Константы модуля* | Константы модуля |
| *Поля дескриптора* | Поля дескриптора |

`popen.shell(`*command*[, *mode*]`)`

Выполнение shell-команды.

**Параметры**

- `command` (*string*) – имя выполняемой команды, обязательно
- `mode` (*string*) – режим передачи данных, необязательно

**возвращает** (при успешном выполнении) дескриптор объекта popen, который мы будем называть `popen_handle` или `ph`

(при неудачном выполнении) `nil, err`

Возможные ошибки: если один из параметров задан некорректно, функция возвращает IllegalParams: неправильно задан тип или значение параметра. Другие возможные ошибки смотрите в разделе *popen.new()*.

Возможные значения режима передачи данных `mode`:

- `'w'` which enables *popen_handle:write()*
- `'r'` which enables *popen_handle:read()*
- `'R'` which enables *popen_handle:read({stderr = true})*
- `'nil'` which means inherit parent's std* file descriptors

Several mode characters can be set together, for example `'rw'`, `'rRw'`.

Функция `shell` — это сокращение для *popen.new({command}, opts)* с `opts.shell.setsid` и `opts.shell.group_signal`, установленными в `true`, и значениями `opts.stdin`, `opts.stdout` и `opts.stderr`, установленными на основе параметра `mode`.

All std* streams are inherited from the parent by default unless it is changed using mode: `'r'` for stdout, `'R'` for stderr, or `'w'` for stdin.

**Пример:**

This is the equivalent of the `sh -c date` command. It starts a process, runs `'date'`, reads the output, and closes the popen object (`ph`).

```
local popen = require('popen')
-- Запуск программы и сохранение ее дескриптора.
local ph = popen.shell('date', 'r')
-- Считывание вывода программы и удаление следующей строки.
local date = ph:read():rstrip()
-- Освобождение ресурсов. Процесс принудительно завершается (но 'date'
-- все равно завершает работу).
ph:close()
print(date)
```

Unix defines a text file as a sequence of lines. Each line is terminated by a newline (\\n) symbol. The same convention is usually applied for text output of a command. So, when it is redirected to a file, the file will be correct.

Однако внутри приложение обычно работает со строками, которые *не* завершаются символом новой строки (например, строки сообщений об ошибках). Символ новой строки обычно добавляется прямо перед записью строки в stdout, консоль или лог. Поэтому в примере выше был использован метод `rstrip()`.

`popen.new(`*argv*`[, `*opts*`])`

Выполнение дочерней программы в новом процессе.

**Параметры**

- `argv` (`array`) – массив, состоящий из запускаемой программы и параметров командной строки, обязательный аргумент; если поле `opts.shell` установлено в false (по умолчанию), то необходимо задать абсолютный путь к программе
- `opts` (*table*) – таблица параметров, необязательно

**возвращает** (при успешном выполнении) дескриптор объекта popen, который мы будем называть `popen_handle` или `ph`

(при неудачном выполнении) `nil, err`

Возможные ошибки:

- IllegalParams: некорректный тип или значение параметра
- IllegalParams: групповой сигнал задан, а setsid — нет

Возможные причины ошибок, когда возвращается `nil, err`:

- SystemError: dup(), fcntl(), pipe(), vfork() или close() завершились с ошибкой в родительском процессе
- SystemError: (временное ограничение) родительский процесс закрыл stdin, stdout или stderr
- OutOfMemory: невозможно выделить память для дескриптора или временного буфера

Возможные элементы `opts`:

- `opts.stdin` (действие над STDIN_FILENO)
- `opts.stdout` (действие над STDOUT_FILENO)
- `opts.stderr` (действие над STDERR_FILENO)

Возможные действия файлового дескриптора в таблице `opts`:

- `popen.opts.INHERIT` (== `'inherit'`) [default] inherit the fd from the parent
- `popen.opts.DEVNULL` (== `'devnull'`) open /dev/null on the fd
- `popen.opts.CLOSE` (== `'close'`) close the fd
- `popen.opts.PIPE` (== `'pipe'`) feed data from fd to parent, or from parent to fd, using a pipe

Таблица `opts` может содержать таблицу `env` переменных среды для использования внутри процесса. Каждый элемент `opts.env` может быть парой ключ-значение (где ключ — имя переменной, а значение — значение переменной).

- Если `opts.env` не установлено, то наследуется текущая среда.
- Если `opts.env` является пустой таблицей, то среда будет сброшена.
- Если `opts.env` является непустой таблицей, то среда будет заменена.

Таблица `opts` может содержать следующие элементы типа boolean:

| Имя | Значение по умолчанию | Назначение |
|---|---|---|
| opts.shell | false | При значении true выполняется запуск дочернего процесса через `sh -c "${opts.argv}"`. При значении false исполняемый процесс вызывается напрямую. |
| opts.setsid | false | При значении true программа запускается в новой сессии. При значении false программа запускается в сессии и группе процессов экземпляра Tarantool. |
| opts.close_fds | true | При значении true закрываются все унаследованные родительские файловые дескрипторы. При значении false унаследованные родительские файловые дескрипторы не закрываются. |
| opts.restore_signals | true | При значении true сбрасываются все действия сигналов, измененные в родительском процессе. При значении false все действия сигналов, измененные в родительском процессе, наследуются. |
| opts.group_signal | false | При значении true отправляется сигнал в группу дочерних процессов, но только при условии, что установлено поле `opts.setsid`. При значении false сигнал отправляется только одному дочернему процессу. |
| opts.keep_child | false | При значении true дочернему процессу (или группе процессов, если поле `opts.group_signal` установлено в true) не отправляется сигнал SIGKILL. При значении false дочернему процессу (или группе процессов, если поле `opts.group_signal` установлено в true) отправляется сигнал SIGKILL при выполнении *popen_handle:close()* или когда Lua GC собирает дескрипторы. |

Возвращаемый дескриптор `ph` дает доступ к методу *popen_handle:close()* для явного освобождения всех занятых ресурсов, включая сам дочерний процесс, если не установлено поле `opts.keep_child`. Однако, если метод `close()` не вызывается для дескриптора в течение его жизни, Lua GC запустит то же самое действие по освобождению ресурсов.

Tarantool рекомендует использовать `opts.setsid` вместе с `opts.group_signal`, если дочерний процесс может создать собственные дочерние процессы и их все нужно будет завершить одновременно.

Сигнал не будет отправлен, если дочерний процесс уже был завершен. В противном случае мы можем случайно завершить другой процесс, который имеет тот же идентификатор процесса после освобождения его предыдущим. Это означает, что если дочерний процесс завершается до того, как завершаются его дочерние процессы, то функция не будет отправлять сигнал группе процессов, даже когда установлены `opts.setsid` и `opts.group_signal`.

Используйте *os.environ()*, чтобы передать копию текущей среды с несколькими заменами (см. Пример 2 ниже).

**Пример 1**

В этом примере выполняется аналог команды `sh -c date`. Происходит запуск процесса, выполняется `'date'`, считывается результат и объект popen (`ph`) закрывается.

```
local popen = require('popen')

local ph = popen.new({'/bin/date'}, {
    stdout = popen.opts.PIPE,
})
```

(continues on next page)

(продолжение с предыдущей страницы)

```
local date = ph:read():rstrip()
ph:close()
print(date) -- например, Thu 16 Apr 2020 01:40:56 AM MSK
```

**Пример 2**

Example 2 is quite similar to Example 1, but sets an environment variable and uses the shell builtin `'echo'` to show it.

```
local popen = require('popen')
local env = os.environ()
env['FOO'] = 'bar'
local ph = popen.new({'echo "${FOO}"'}, {
    stdout = popen.opts.PIPE,
    shell = true,
    env = env,
})
local res = ph:read():rstrip()
ph:close()
print(res) -- bar
```

**Пример 3**

Пример 3 показывает, как перехватить дочерний поток stderr.

```
local popen = require('popen')
local ph = popen.new({'echo hello >&2'}, { -- !!
    stderr = popen.opts.PIPE,              -- !!
    shell = true,
})
local res = ph:read({stderr = true}):rstrip()
ph:close()
print(res) -- hello
```

**Пример 4**

Пример 4 показывает, как запустить потоковую программу (например, `grep`, `sed` и т.д.), записать данные в ее поток stdin и считать данные из stdout.

В этом примере предполагается, что входные данные достаточно малы, чтобы поместиться в буфер канала (обычно его размер равен 64 КиБ, но это зависит от конкретной платформы и ее конфигурации).

Если процесс записывает большое количество данных, он приостановится при выполнении *popen_handle:write()*. Для разрешения этой проблемы вызывайте *popen_handle:read()* в цикле в другом файбере (запустите его перед первым вызовом `:write()`).

Если процесс записывает длинный текст в stderr, он может приостановиться при выполнении `write()`, потому что буфер канала stderr заполнился. Для решения этой проблемы считывайте из stderr в отдельном файбере.

```
local function call_jq(input, filter)
    -- Запуск процесса jq, соединение с stdin, stdout и stderr.
    local jq_argv = {'/usr/bin/jq', '-M', '--unbuffered', filter}
    local ph, err = popen.new(jq_argv, {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        stdin = popen.opts.PIPE,
        stdout = popen.opts.PIPE,
        stderr = popen.opts.PIPE,
    })
    if ph == nil then return nil, err end
    -- Запись входных данных в дочерний stdin и отправка EOF.
    local ok, err = ph:write(input)
    if not ok then return nil, err end
    ph:shutdown({stdin = true})
    -- Считывание всех данных до EOF.
    local chunks = {}
    while true do
        local chunk, err = ph:read()
        if chunk == nil then
            ph:close()
            return nil, err
        end
        if chunk == '' then break end -- EOF
        table.insert(chunks, chunk)
    end
    -- Считывание данных диагностики из stderr (при наличии).
    local err = ph:read({stderr = true})
    if err ~= '' then
        ph:close()
        return nil, err
    end
    -- Соединение всех частей вместе, обрезка символа конца строки.
    return table.concat(chunks):rstrip()
end
```

#### методы дескриптора popen

`object popen_handle`

`popen_handle:read(`[*opts*]`)`

Считывание данных из дочернего процесса.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*
- `opts` (*table*) – параметры

Возможные элементы `opts`:

- `opts.stdout` (boolean, значение по умолчанию — `true`, при значении `true` выполняется считывание из stdout)
- `opts.stderr` (boolean, значение по умолчанию — `false`, при значении `true` выполняется считывание из stderr)
- `opts.timeout` (число, значение по умолчанию — 100 лет, временная квота в секундах)

In other words: by default `read()` reads from stdout, but reads from stderr if one sets `opts.stderr` to `true`. It is not legal to set both `opts.stdout` and `opts.stderr` to `true`.

**возвращает** (при успешном выполнении) строка со считанным значением, пустая строка при EOF

(при неудачном выполнении) `nil, err`

**Possible errors**

These errors are raised on incorrect parameters or when the fiber is cancelled:

- IllegalParams: некорректный тип или значение параметра
- IllegalParams: вызов дескриптора, который уже был закрыт
- IllegalParams: одновременно установлены opts.stdout и opts.stderr
- IllegalParams: запрашиваемая операция ввода-вывода не поддерживается дескриптором (вывод stdout / stderr не перенаправлен)
- IllegalParams: попытка произвести операцию над закрытым файловым дескриптором
- FiberIsCancelled: cancelled by external code

`nil, err` is returned on following failures:

- SocketError: ошибка ввода-вывода при выполнении read()
- TimedOut: превышение квоты opts.timeout
- OutOfMemory: недостаточно памяти для считывания в буфер
- LuajitError: («not enough memory»): недостаточно памяти для строки Lua

`popen_handle:write(`*str*[, *opts* ]`)`

Запись строки `str` в поток stdin дочернего процесса.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*
- `str` (*string*) – строка для записи
- `opts` (*table*) – параметры

**возвращает** `true` on success, `false` on error

**возвращаемое значение** (при успешном выполнении) boolean = true

(при неудачном выполнении) `nil, err`

Possible `opts` items are: `opts.timeout` (number, default 100 years, time quota in seconds).

Возможные ошибки:

- IllegalParams: некорректный тип или значение параметра
- IllegalParams: вызов дескриптора, который уже был закрыт
- IllegalParams: длина строки превышает SSIZE_MAX
- IllegalParams: запрашиваемая операция ввода-вывода не поддерживается дескриптором (вывод stdin не перенаправлен)
- IllegalParams: попытка произвести операцию над закрытым файловым дескриптором
- FiberIsCancelled: файбер отменен во внешней программе

Возможные причины ошибок, когда возвращается `nil, err`:

- SocketError: ошибка ввода-вывода при выполнении write()
- TimedOut: превышение квоты opts.timeout

`write()` может передать управление (yield) и заблокировать файбер, если дочерний процесс не считывает данные из stdin и буфер канала заполнился. Размер буфера зависит от платформы. Если сомневаетесь, обратите внимание на опцию `opts.timeout`.

Когда опция `opts.timeout` не установлена, `write()` блокирует файбер до момента полной записи данных или возникновения ошибки записи.

`popen_handle:shutdown([opts])`

Закрытие канала с std* со стороны родителя.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*
- `opts` (*table*) – параметры

**возвращает** `true` on success, `false` on error

**возвращаемое значение** (при успешном выполнении) boolean = true

Возможные элементы `opts`:

- `opts.stdin` (boolean) закрыть stdin со стороны родителя
- `opts.stdout` (boolean) закрыть stdout со стороны родителя
- `opts.stderr` (boolean) закрыть stderr со стороны родителя

Мы можем использовать термин std* для указания на любой из этих элементов.

Возможные ошибки:

- IllegalParams: некорректный параметр дескриптора
- IllegalParams: вызов дескриптора, который уже был закрыт
- IllegalParams: не выбран ни один из потоков stdin, stdout или stderr
- IllegalParams: запрашиваемая операция ввода-вывода не поддерживается дескриптором (один из потоков std* не перенаправлен)

Основная цель использования `shutdown()` — отправка EOF в дочерний поток stdin. Однако stdout / stderr может быть уже закрыт со стороны родительского процесса.

`shutdown()` does not fail on already closed fds (idempotence). However, it fails on an attempt to close the end of a pipe that never existed. In other words, only those `std*` options that were set to `popen.opts.PIPE` during handle creation may be used here (for *popen.shell()*: `'r'` corresponds to stdout, `'R'` to stderr and `'w'` to stdin).

`shutdown()` не закрывает никакие файловые дескрипторы при завершении с ошибкой: либо закрываются все запрашиваемые дескрипторы (при успешном выполнении), либо ни один из них.

**Пример:**

```
local popen = require('popen')
local ph = popen.shell('sed s/foo/bar/', 'rw')
ph:write('lorem foo ipsum')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
ph:shutdown({stdin = true})
local res = ph:read()
ph:close()
print(res) -- lorem bar ipsum
```

`popen_handle:terminate()`
Отправка сигнала SIGTERM дочернему процессу.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*

**возвращает** для получения информации об ошибках и возвращаемых значениях смотрите *popen_ handle:signal()*

`terminate()` просто отправляет сигнал SIGTERM. Он *не* освобождает никакие ресурсы (такие как память для дескрипторов popen и файловые дескрипторы).

`popen_handle:kill()`
Отправка сигнала SIGKILL дочернему процессу.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*

**возвращает** для получения информации об ошибках и возвращаемых значениях смотрите *popen_ handle:signal()*

`kill()` просто отправляет сигнал SIGKILL. Он *не* освобождает никакие ресурсы (такие как память для дескрипторов popen и файловые дескрипторы).

`popen_handle:signal(`*signo*`)`
Отправка сигнала дочернему процессу.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*
- `signo` (`number`) – отправляемый сигнал

**возвращает** (при успешном выполнении) `true` (сигнал отправлен)

(при неудачном выполнении) `nil, err`

Возможные ошибки:

- IllegalParams: некорректный параметр дескриптора
- IllegalParams: вызов дескриптора, который уже был закрыт

Возможные значения ошибок при возвращении `nil, err`:

- SystemError: процесс больше не существует (также может возвращаться для зомби-процессов или когда все процессы в группе являются зомби-процессами (но см. примечание для Mac OS ниже)
- SystemError: неправильный номер сигнала
- SystemError: нет разрешения на отправку сигнала процессу или группе процессов (возвращается на Mac OS, когда сигнал отправляется группе процессов, где лидер группы

является зомби-процессом (или все процессы являются зомби-процессами, детали неясны) (эта ошибка также может возникнуть по другим причинам, детали неясны)

Если для дескриптора установлены `opts.setsid` и `opts.group_signal`, сигнал отправляется группе процессов, а не отдельному процессу. Для подробной информации по групповым сигналам смотрите *popen.new()*. Внимание: на Mac OS процесс в группе может не получить сигнал, особенно если он только что был разветвлен (возможно это происходит из-за состояния гонки).

Примечание: Некоторые сигналы имеют разные номера на разных платформах. Поэтому в этом модуле мы предлагаем константы `popen.signal.SIG*`.

`popen_handle:info()`

Получение информации о дескрипторе popen.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*
- `signo` (`number`) – отправляемый сигнал

**возвращает** (при успешном выполнении) отформатированный результат

**возвращаемое значение** res

Возможные ошибки:

- IllegalParams: некорректный параметр дескриптора
- IllegalParams: вызов дескриптора, который уже был закрыт

Результат выводится в следующем формате:

```
{
    pid = <number> or <nil>,
    command = <string>,
    opts = <table>,
    status = <table>,
    stdin = one-of(
        popen.stream.OPEN   (== 'open'),
        popen.stream.CLOSED (== 'closed'),
        nil,
    ),
    stdout = one-of(
        popen.stream.OPEN   (== 'open'),
        popen.stream.CLOSED (== 'closed'),
        nil,
    ),
    stderr = one-of(
        popen.stream.OPEN   (== 'open'),
        popen.stream.CLOSED (== 'closed'),
        nil,
    ),
}
```

`pid` — это идентификатор процесса, когда тот находится в рабочем состоянии; для завершенного процесса `pid` имеет значение nil.

`command` — это конкатенация аргументов, разделенных пробелами, которые были переданы в `execve()`. Аргументы, состоящие из нескольких слов, заключаются в кавычки. Кавычки внутри аргументов не экранируются.

`opts` – это таблица параметров дескриптора, описанная в разделе `opts` функции *popen.new()*. `opts.env` здесь не отображается, потому что карта переменных среды не хранится в дескрипторе.

`status` — это таблица, отображающая состояние процесса в следующем формате:

```
{
    state = one-of(
        popen.state.ALIVE    (== 'alive'),
        popen.state.EXITED   (== 'exited'),
        popen.state.SIGNALED (== 'signaled'),
    )
    -- Отображается при состоянии процесса 'завершенный'.
    exit_code = <number>,
    -- Отображается при состоянии процесса 'принимающий сигнал'.
    signo = <number>,
    signame = <string>,
}
```

`stdin`, `stdout`, and `stderr` reflect the status of the parent's end of a piped stream. If a stream is not piped, the field is not present (`nil`). If it is piped, the status may be either `popen.stream.OPEN` (== `'open'`) or `popen.stream.CLOSED` (== `'closed'`). The status may be changed from `'open'` to `'closed'` by a *popen_ handle:shutdown({std... = true})* call.

**Пример 1**

(в консоли Tarantool)

```
tarantool> require('popen').new({'/usr/bin/touch', '/tmp/foo'})
---
- command: /usr/bin/touch /tmp/foo
  status:
    state: alive
  opts:
    stdout: inherit
    stdin: inherit
    group_signal: false
    keep_child: false
    close_fds: true
    restore_signals: true
    shell: false
    setsid: false
    stderr: inherit
  pid: 9499
...
```

**Пример 2**

(в консоли Tarantool)

```
tarantool> require('popen').shell('grep foo', 'wrR')
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- stdout: open
  command: sh -c 'grep foo'
  stderr: open
  status:
    state: alive
  stdin: open
  opts:
    stdout: pipe
    stdin: pipe
    group_signal: true
    keep_child: false
    close_fds: true
    restore_signals: true
    shell: true
    setsid: true
    stderr: pipe
  pid: 10497
...
```

`popen_handle:wait()`

Ожидание, пока дочерний процесс не завершится или не получит сигнал.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*
- `signo` (`number`) – отправляемый сигнал

**возвращает** (при успешном выполнении) отформатированный результат

**возвращаемое значение** res

Возможные ошибки:

- IllegalParams: некорректный параметр дескриптора
- IllegalParams: вызов дескриптора, который уже был закрыт
- FiberIsCancelled: файбер отменен во внешней программе

Отформатированный результат представляет собой таблицу состояний процесса (аналогично компоненту `status` таблицы, возвращаемой через *popen_ handle:info()*).

`popen_handle:close()`

Закрытие дескриптора popen.

**Параметры**

- `ph` (`handle`) – дескриптор дочернего процесса, созданный через *popen.new()* или *popen.shell()*

**возвращает** (при успешном выполнении) true

(при неудачном выполнении) `nil, err`

Возможные ошибки:

- IllegalParams: некорректный параметр дескриптора

Возможные результаты диагностики, когда возвращается `nil, err` (не рассматривайте эти случаи как ошибки):

- SystemError: нет разрешения на отправку сигнала процессу или группе процессов (это сообщение диагностики может появиться из-за особенностей обработки зомби-процессов в Mac OS, когда установлен `opts.group_signal`, см. *popen_handle:signal()*. Оно также может появиться по другим причинам, детали неясны).

Если известно, что процесс был завершен, то в результате всегда возвращается `true` (например, после выполнения *popen_handle:wait()* не будет отправлено никакого сигнала, так что никакой ошибки не может возникнуть).

`close()` принудительно завершает процесс через SIGKILL и освобождает все ресурсы, связанные с дескриптором popen.

Подробная информация об отправке сигналов:

- Сигнал отправляется только когда поле opts.keep_child не установлено.
- Сигнал отправляется только когда процесс находится в рабочем состоянии согласно информации, доступной на текущей итерации цикла событий. (Здесь есть слабое место: сигнал может быть отправлен зомби-процессу, но это не представляет никакой угрозы).
- Сигнал отправляется процессу или группе процессов в зависимости от `opts.group_signal`. (Для подробной информации о групповых сигналах смотрите *popen.new()*).

Ресурсы освобождаются вне зависимости от того, успешно ли отправился сигнал: дескрипторы файлов закрываются, память освобождается, а дескриптор popen помечается как закрытый.

Над закрытым дескриптором невозможно выполнять никакие операции кроме `close()`, которая всегда выполняется успешно над закрытым дескриптором (идемпотентность).

`close()` может вернуть `true` или `nil, err`, но она всегда освобождает ресурсы дескриптора. Поэтому для того, кто отправил сигнал, любое возвращаемое значение означает успешное выполнение. Возвращаемые значения только дают информацию для логирования или составления отчетов.

**Поля дескриптора**

```
popen_handle.pid
popen_handle.command
popen_handle.opts
popen_handle.status
popen_handle.stdin
popen_handle.stdout
popen_handle.stderr
```

За более подробной информацией обратитесь к *popen_handle:info()*.

**Константы модуля**

```
- popen.opts
  - INHERIT (== 'inherit')
  - DEVNULL (== 'devnull')
  - CLOSE   (== 'close')
  - PIPE    (== 'pipe')

- popen.signal
  - SIGTERM (== 9)
  - SIGKILL (== 15)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  - ...

- popen.state
  - ALIVE    (== 'alive')
  - EXITED   (== 'exited')
  - SIGNALED (== 'signaled')

- popen.stream
  - OPEN    (== 'open')
  - CLOSED  (== 'closed')
```

### 9.5.29 Модуль socket

#### Общие сведения

Модуль `socket` позволяет обмениваться данными с локальным или удаленным хостом по BSD-сокетам в режиме с установлением соединений (TCP) или на основе датаграмм (UDP). Семантика вызовов в API модуля `socket` точно соответствует семантике соответствующих вызовов в POSIX.

Функции для настройки и подключения: `socket`, `sysconnect`, `tcp_connect`. Функции для отправки данных: `send`, `sendto`, `write`, `syswrite`. Функции для получения данных: `recv`, `recvfrom`, `read`. Функции для ожидания отправки/получения данных: `wait`, `readable`, `writable`. Функции для установки флагов: `nonblock`, `setsockopt`. Функции для остановки и отключения: `shutdown`, `close`. Функции для проверки ошибок: `errno`, `error`.

#### Указатель

Ниже приведен перечень всех функций модуля `socket`.

| Имя | Назначение |
|---|---|
| *socket()* | Создание сокета |
| *socket.tcp_connect()* | Подключение к удаленному хосту с помощью сокета |
| *socket.getaddrinfo()* | Получение информации об удаленном узле |
| *socket.tcp_server()* | Использование Tarantool в качестве TCP-сервера |
| *socket.bind()* | Привязка сокета к данному хосту/порту |
| *socket_object:sysconnect()* | Подключение к удаленному хосту с помощью сокета |
| *socket_object:send()*<br>*socket_object:write()* | Отправка данных по подключенному сокету |
| *socket_object:syswrite()* | Запись данных в буфер сокета без блокировки |
| *socket_object:recv()* | Чтение с подключенного сокета |
| *socket_object:sysread()* | Чтение данных из буфера сокета без блокировки |
| *socket_object:bind()* | Привязка сокета к данному хосту/порту |
| *socket_object:listen()* | Начало прослушивания входящих соединений |
| *socket_object:accept()* | Принятие запроса клиента на соединение + создание подключенного сокета |
| *socket_object:sendto()* | Отправка сообщения по UDP-сокету на указанный хост |
| *socket_object:recvfrom()* | Получение сообщения по UDP-сокету |
| *socket_object:shutdown()* | Отключение передачи данных на чтение, на запись или в обоих направлениях |
| *socket_object:close()* | Закрытие сокета |
| *socket_object:error()*<br>*socket_object:errno()* | Получение информации о последней ошибке на сокете |
| *socket_object:setsockopt()* | Определение флагов сокета |
| *socket_object:getsockopt()* | Получение флагов сокета |
| *socket_object:linger()* | Установить/убрать флаг SO_LINGER |
| *socket_object:nonblock()* | Определить/получить значение флага |
| *socket_object:readable()* | Ожидание доступности чего-либо для чтения |
| *socket_object:writable()* | Ожидание доступности чего-либо для записи |
| *socket_object:wait()* | Ожидание доступности чего-либо для чтения или записи |
| *socket_object:name()* | Получение информации о ближней стороне соединения |
| *socket_object:peer()* | Получение информации о дальней стороне соединения |
| *socket.iowait()* | Ожидание активности чтения/записи |
| *LuaSocket wrapper functions* | Несколько методов эмуляции LuaSocket API |

Typically a socket session will begin with the setup functions, will set one or more flags, will have a loop with sending and receiving functions, will end with the teardown functions – as an example at the end of this section will show. Throughout, there may be error-checking and waiting functions for synchronization. To prevent a fiber containing socket functions from «blocking» other fibers, the *implicit yield rules* will cause a yield so that other processes may take over, as is the norm for *cooperative multitasking*.

Для всех примеров в данном разделе имя сокета будет sock, а вызов функции будет выглядеть как `sock:имя_функции(...)`.

`socket.__call(`*domain*, *type*, *protocol*`)`

Создание нового TCP-сокета или UDP-сокета. Значения аргумента остаются теми же, что и на странице socket(2) руководства по Linux.

**возвращает** неподключенный сокет или nil.

**тип возвращаемого значения** пользовательские данные

**Пример:**

```
socket('AF_INET', 'SOCK_STREAM', 'tcp')
```

socket.tcp_connect(*host*[, *port*[, *timeout*]])

Подключение к удаленному хосту с помощью сокета.

**Параметры**

- host (*string*) – URL или IP-адрес
- port (number) – номер порта
- timeout (number) – количество секунд ожидания

**возвращает** (if error) {nil, error-message-string}. (if no error) a new socket object.

**тип возвращаемого значения** socket object, which may be viewed as a table

**Пример:**

```
sock, e = socket.tcp_connect('127.0.0.1', 3301)
if sock == nil then print(e) end
```

socket.getaddrinfo(*host*, *port*[, *timeout*[, *{option-list}*]])

socket.getaddrinfo(*host*, *port*[, *{option-list}*])

Функция socket.getaddrinfo() используется для поиска информации об удаленном узле, чтобы можно было передать правильные аргументы для sock:sysconnect(). Эта функция может использовать конфигурационный параметр *worker_pool_threads*.

**Параметры**

- host (*string*) – URL или IP-адрес
- port (number/string) – номер порта — число или строка
- timeout (number) – количество секунд ожидания
- options (*table*) –
  - type – предпочтительный тип сокета
  - family – предпочтительное семейство адресов
  - protocol
  - flags – дополнительные опции (подробнее о них здесь)

**возвращает** (if error) {nil, error-message-string}. (if no error) A table containing these fields: «host», «family», «type», «protocol», «port».

**тип возвращаемого значения** таблица

**Пример:**

```
tarantool> socket.getaddrinfo('tarantool.org', 'http')
---
- - host: 188.93.56.70
    family: AF_INET
    type: SOCK_STREAM
    protocol: tcp
    port: 80
  - host: 188.93.56.70
    family: AF_INET
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    type: SOCK_DGRAM
    protocol: udp
    port: 80
...
-- To find the available values for the options use the following:
tarantool> socket.internal.AI_FLAGS -- or SO_TYPE, or DOMAIN
---
- AI_ALL: 256
  AI_PASSIVE: 1
  AI_NUMERICSERV: 4096
  AI_NUMERICHOST: 4
  AI_V4MAPPED: 2048
  AI_ADDRCONFIG: 1024
  AI_CANONNAME: 2
...
```

socket.tcp_server(*host*, *port*, *handler-function-or-table*[, *timeout*])

Функция socket.tcp_server() заставляет Tarantool выступать в качестве сервера для принятия подключений. Обычно для этой же цели используется *box.cfg{listen=...}*.

**Параметры**

- host (*string*) – имя или IP хоста
- port (number) – порт хоста, может быть 0
- handler-function-or-table (function/table) – что выполнить после подключения
- timeout (number) – время ожидания в секундах при разрешении имени хоста в IP-адрес

**возвращает** (if error) {nil, error-message-string}. (if no error) a new socket object.

**тип возвращаемого значения** socket object, which may be viewed as a table

Параметр handler-function-or-table может представлять собой просто имя функции или объявление функции: handler_function. Или же может быть таблицей: {handler = *handler_function* [, prepare = *prepare_function*] [, name = *name*] }. Функция handler_function является обязательной, в ней может быть только один параметр = сокет (используется для непрерывной работы после установки соединения), выполняется один раз за соединение после того, как произойдет accept(). Функция prepare_function необязательна; она выполняется однократно перед установкой соединения (bind()) на слушающем сокете и должна возвращать либо значение бэклога, либо ничего. Например:

```
socket.tcp_server('localhost', 3302, function (s) loop_loop() end)
socket.tcp_server('localhost', 3302, {handler=hfunc, name='name'})
socket.tcp_server('localhost', 3302, {handler=hfunc, prepare=pfunc})
```

Более полный пример см. в разделе *Использование tcp_server для получения содержимого файла, отправленного по socat* и *Использование tcp_server с handler и prepare*.

socket.bind(*host*, *port*)

Bind a socket to the given host/port. This is equivalent to *socket_object:bind()*, but is done on the result of require('socket'), rather than on the socket object.

**Параметры**

- host (*string*) – URL или IP-адрес
- port (number) – номер порта

**возвращает** (if error) {nil, error-message-string}. (if no error) A table which may have information about the bind result.

**тип возвращаемого значения** таблица

object socket_object

socket_object:sysconnect(*host*, *port*)

Подключение к удаленному хосту с помощью существующего сокета. Значения аргументов будут такие же, как в *tcp_connect()*. Хост должен представлять собой IP-адрес.

**Параметры:**

- **Либо:**
  - host – строковое представление IPv4 адреса или IPv6 адреса;
  - port – число.
- **Либо:**
  - host – строка, которая содержит «unix/»;
  - port – строка, которая содержит путь к Unix-сокету.
- **Либо:**
  - host – число, 0 (ноль), что означает «все локальные интерфейсы»;
  - port – число. Если номер порта – 0 (ноль), сокет будет привязан к случайному локальному порту.

**возвращает** значение объекта сокета может изменяться, если будет выполнена функция sysconnect().

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
socket = require('socket')
sock = socket('AF_INET', 'SOCK_STREAM', 'tcp')
sock:sysconnect(0, 3301)
```

socket_object:send(*data*)
socket_object:write(*data*)

Отправка данных по подключенному сокету.

**Параметры**

- data (*string*) – что отправляется

**возвращает** количество отправляемых байтов.

**тип возвращаемого значения** число

Возможные ошибки: nil в случае ошибки.

socket_object:syswrite(*size*)

Запись максимально возможного количества данных в буфер сокета без блокировки. Используется редко. Для получения подробной информации см. описание.

`socket_object:recv(`*size*`)`

Чтение количества байтов, определенного в `size`, из подключенного сокета. Внутренний буфер опережающего считывания используется для уменьшения использования ресурсов на вызов.

**Параметры**

- `size` (`integer`) – максимальное количество получаемых байтов. См. *Рекомендованный размер*.

**возвращает** строка запрошенной длины, если выполнено.

**тип возвращаемого значения** строка

Возможные ошибки: В случае ошибки возвращается пустая строка, после чего статус, errno, errstr. Если передача данных на запись закрыта с другой стороны, возвращаются оставшиеся для чтения данные из сокета (возможно, пустая строка), после чего идет статус «eof» (конец файла).

`socket_object:read(`*limit*[, *timeout*]`)`

`socket_object:read(`*delimiter*[, *timeout*]`)`

`socket_object:read(`*{options}*[, *timeout*]`)`

Чтение данных из подключенного сокета до выполнения какого-либо условия и возврат прочтенных байтов. Производится чтения количества байтов, которое указано в параметре `limit`, либо до символа-разделителя, либо до истечения времени ожидания. В отличие от `socket_object:recv` (где используется внутренний буфер опережающего считывания), `socket_object:read` зависит от буфера сокета.

**Параметры**

- `limit` (`integer`) – максимальное количество байтов для чтения, например, 50 означает «остановиться на 50 байтах»
- `delimiter` (*`string`*) – separator for example `?` means «stop after a question mark»; this parameter can accept a table of separators, for example, `delimiter = {"\n", "\r"}`
- `timeout` (`number`) – максимальное количество секунд ожидания, например, 50 означает «остановиться через 50 секунд».
- `options` (*`table`*) – `chunk=`*`предел`* и/или `delimiter=`*`разделитель`*, например, `{chunk=5,delimiter='x'}`.

**возвращает** пустая строка, если нет данных для чтения, либо нулевое значение nil в случае ошибки, либо строка, ограниченная количеством байтов в `limit`, которая может включать в себя байты, совпадающие с выражением `delimiter`.

**тип возвращаемого значения** строка

`socket_object:sysread(`*size*`)`

Возврат данных из буфера сокета без блокировки.Если сокет с блокировкой, `sysread()` может блокировать процесс вызова. Используется редко. Для получения подробной информации, см. описание.

**Параметры**

- `size` (`integer`) – максимальное количество байтов для чтения, например, 50 означает «остановиться на 50 байтах»

**возвращает** пустая строка, если нет данных для чтения, либо нулевое значение nil в случае ошибки, либо строка, ограниченная количеством байтов в `size`.

**тип возвращаемого значения** строка

socket_object:bind(*host*[, *port*])
Привязка сокета к данному хосту/порту. UDP-сокет после привязки может использоваться для получения данных (см. *socket_object.recvfrom*). TCP-сокет может использоваться для принятия новых соединений после перевода в режим прослушивания.

**Параметры**

- host (*string*) – URL или IP-адрес
- port (number) – номер порта

**возвращает** true (правда), если выполнено, false (ложь) в случае ошибки. Если возвращается false, используйте *socket_object:errno()* или *socket_object:error()* для получения подробной информации.

**тип возвращаемого значения** boolean (логический)

socket_object:listen(*backlog*)
Начало прослушивания входящих соединений.

**Параметры**

- backlog – в Linux очередь запросов backlog может быть в /proc/sys/net/core/somaxconn, в BSD очередь запросов может представлять собой SOMAXCONN.

**возвращает** true (правда), если выполнено, false (ложь) в случае ошибки.

**тип возвращаемого значения** boolean (логический).

socket_object:accept()
Принятие нового клиентского соединения и создание нового подключенного сокета. Установка блокирующего режима на сокете явным образом после принятия соединения приведет к эффективной работе.

**возвращает** новый сокет, если выполнено.

**тип возвращаемого значения** пользовательские данные

Возможные ошибки: nil.

socket_object:sendto(*host*, *port*, *data*)
Отправка сообщения по UDP-сокету на указанный хост.

**Параметры**

- host (*string*) – URL или IP-адрес
- port (number) – номер порта
- data (*string*) – что отправляется

**возвращает** количество отправляемых байтов.

**тип возвращаемого значения** число

Возможные ошибки: в случае ошибки возвращает nil, а также может вернуть статус, errno, errstr.

socket_object:recvfrom(*size*)
Получение сообщения по UDP-сокету.

**Параметры**

- `size` (`integer`) – максимальное количество получаемых байтов. См. *Рекомендованный размер*.

**возвращает** сообщение, таблица с полями «host», «family» и «port».

**тип возвращаемого значения** строка, таблица

Возможные ошибки: в случае ошибки возвращает nil, статус, errno, errstr.

**Пример:**

После `message_content, message_sender = recvfrom(1)` значением `message_content` может быть строка, которая содержит „X“, а значением `message_sender` может быть таблица, которая содержит

```
message_sender.host = '18.44.0.1'
message_sender.family = 'AF_INET'
message_sender.port = 43065
```

`socket_object:shutdown(`*how*`)`

Отключение передачи данных на чтение, на запись или в обоих направлениях.

**Параметры**

- `how` – socket.SHUT_RD, socket.SHUT_WR, or socket.SHUT_RDWR.

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

`socket_object:close()`

Закрытие (удаление) сокета. Закрытый сокет больше не должен использоваться. Сокет будет закрыт автоматически, когда сборщик мусора Lua удалит данные.

**возвращает** true (правда), если выполнено, false (ложь) в случае ошибки. Например, если сокет sock уже закрыт, sock:close() вернет false.

**тип возвращаемого значения** boolean (логический)

`socket_object:error()`
`socket_object:errno()`

Получение информации о последней ошибке на сокете, если таковая была. Ошибки не выдают исключения, поэтому данные функции необходимы.

**возвращает** результат `sock:errno()`, результат `sock:error()`. Если ошибки нет, то `sock:errno()` вернет 0 и `sock:error()`.

**тип возвращаемого значения** число, строка

`socket_object:setsockopt(`*level*, *name*, *value*`)`

Определение флагов сокета. Значения аргумента будут такими же, что и на странице getsockopt(2) руководства по Linux. Tarantool принимает следующие:

- SO_ACCEPTCONN
- SO_BINDTODEVICE
- SO_BROADCAST
- SO_DEBUG
- SO_DOMAIN
- SO_ERROR
- SO_DONTROUTE

- SO_KEEPALIVE
- SO_MARK
- SO_OOBINLINE
- SO_PASSCRED
- SO_PEERCRED
- SO_PRIORITY
- SO_PROTOCOL
- SO_RCVBUF
- SO_RCVBUFFORCE
- SO_RCVLOWAT
- SO_SNDLOWAT
- SO_RCVTIMEO
- SO_SNDTIMEO
- SO_REUSEADDR
- SO_SNDBUF
- SO_SNDBUFFORCE
- SO_TIMESTAMP
- SO_TYPE

Установка флага SO_LINGER осуществляется с помощью `sock:linger(active)`.

`socket_object:getsockopt(`*level*, *name*`)`
Получение флагов сокета. Список возможных флагов см. с помощью `sock:setsockopt()`.

`socket_object:linger(`[*active*]`)`
Установить или убрать флаг SO_LINGER. Описание флага см. в руководстве по Linux.

**Параметры**

- `active` (`boolean`) –

**возвращает** новые значения active и timeout.

`socket_object:nonblock(`[*flag*]`)`

- `sock:nonblock()` возвращает текущее значение флага.
- `sock:nonblock(false)` устанавливает флаг на false и возвращает false.
- `sock:nonblock(true)` устанавливает флаг на true и возвращает true.

Эту функцию можно использовать до вызова функции, которая в противном случае будет блокировать бесконечно.

`socket_object:readable(`[*timeout*]`)`
Ожидание доступности чего-либо для чтения или до истечения времени ожидания.

**возвращает** true, если сокет доступен для чтения, false, если истекло время ожидания;

`socket_object:writable([timeout])`
Ожидание доступности чего-либо для записи или до истечения времени ожидания.

**возвращает** true, если сокет доступен для записи, false, если истекло время ожидания;

`socket_object:wait([timeout])`
Ожидание доступности чего-либо для чтения или записи, или до истечения времени ожидания.

**возвращает** „R“, если сокет доступен для чтения, „W“, если сокет доступен для записи, „RW“, если сокет доступен и для чтения, и для записи, „“ (пустая строка), если истекло время ожидания;

`socket_object:name()`
Функция `sock:name()` используется для получения информации о ближней стороне соединения. Если сокет привязан к `xyz.com:45`, то `sock:name` вернет информацию о `[host:xyz.com, port:45]`. Аналогичная функция в POSIX – `getsockname()`.

**возвращает** Таблица со следующими полями: «host», «family», «type», «protocol», «port».

**тип возвращаемого значения** таблица

`socket_object:peer()`
Функция `sock:peer()` используется для получения информации о дальней стороне соединения. Если TCP-соединение установлено с удаленным хостом `tarantool.org:80`, то `sock:peer()` вернет информацию о `[host:tarantool.org, port:80]`. Аналогичная функция в POSIX – `getpeername()`.

**возвращает** Таблица со следующими полями: «host», «family», «type», «protocol», «port».

**тип возвращаемого значения** таблица

`socket.iowait(`*fd*, *read-or-write-flags*`[, `*timeout*`])`
Функция `socket.iowait()` используется для ожидания, пока дескриптор файла не будет активен для чтения или записи.

**Параметры**

- `fd` – дескриптор файла
- `read-or-write-flags` – „R“ или 1 = чтение, „W“ или 2 = запись, „RW“ или 3 = чтение|запись.
- `timeout` – количество секунд ожидания

Если значение параметра fd – nil, то будет режим ожидания до истечения времени, указанного в параметре timeout. Если timeout – nil или не указан, время ожидания считается бесконечным.

Как правило, возвращается значение совершенного действия („R“ или „W“, или „RW“, или 1, или 2, или 3). Если время ожидания в timeout проходит без действий чтения или записи, возвращается ошибка = ETIMEDOUT.

Пример: `socket.iowait(sock:fd(), 'r', 1.11)`

### Функции обертки LuaSocket

The LuaSocket API has functions that are equivalent to the ones described above, with different names and parameters, for example `connect()` rather than `tcp_connect()`. Tarantool supports these functions so that third-party packages which depend on them will work.

Проект LuaSocket находится на github. Описание API находится в руководстве по LuaSocket (нажмите на ссылки «введение» и «ссылка» внизу главной страницы руководства).

Пример для Tarantool - *Использование сокета с функциями обертки LuaSocket*.

### Рекомендованный размер

Для `recv` и `recvfrom`: используйте необязательный параметр `size`, чтобы ограничить количество получаемых байтов. Часто используется заданный размер, такой как 512; но во многих случаях лучше использовать предварительно рассчитанный размер, который зависит от контекста – как формат сообщения или состояние сети. Что касается `recvfrom`, следует помнить, что размер больше максимального размера полезного блока данных одного пакета (Maximum Transmission Unit) может вызвать низкоэффективную передачу данных. Что касается Mac OS X, следует отметить, что размер можно настроить с помощью `sysctl net.inet.udp.maxdgram`.

Если размер `size` не задан: Tarantool сделает дополнительный вызов для расчет необходимого количества байтов. Такой дополнительный вызов занимает время, поэтому во избежание низкой эффективности лучше указать `size`.

Если размер `size` задан: в UDP-сокете лишние байты отбрасываются; в TCP-сокете лишние байты не отбрасываются, их можно получить при следующем вызове.

### Примеры

### Использование TCP-сокета в интернете

В данном примере устанавливается соединение по интернету между экземпляром Tarantool и tarantool.org, затем отправляется HTTP-сообщение заголовка «head» и возвращается ответ: «`HTTP/1.1 200 OK`» или что-то другое, если сайт перемещен. Так не слишком удобно взаимодействовать с определенным сайтом, но пример показывает работу системы.

```
tarantool> socket = require('socket')
---
...
tarantool> sock = socket.tcp_connect('tarantool.org', 80)
---
...
tarantool> type(sock)
---
- table
...
tarantool> sock:error()
---
- null
...
tarantool> sock:send("HEAD / HTTP/1.0\r\nHost: tarantool.org\r\n\r\n")
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- 40
...
tarantool> sock:read(17)
---
- HTTP/1.1 302 Move
...
tarantool> sock:close()
---
- true
...
```

### Использование сокета с функциями обертки LuaSocket

Это вариант более раннего примера «Использование TCP-подключения через Интернет». В нем используются *функции обертки LuaSocket*, с слишком коротким временем ожидания, так что, скорее всего, произойдет ошибка «Connection timed out» (Таймаут соединения). Более распространенным способом определения таймаута является использование функции *tcp_connect()*.

```
tarantool> socket = require('socket')
---
...
tarantool> sock = socket.connect('tarantool.org', 80)
---
...
tarantool> sock:settimeout(0.001)
---
- 1
...
tarantool> sock:send("HEAD / HTTP/1.0\r\nHost: tarantool.org\r\n\r\n")
---
- 40
...
tarantool> sock:receive(17)
---
- null
- Connection timed out
...
tarantool> sock:close()
---
- 1
...
```

### Использование UDP-сокета на localhost

Ниже приведен пример с датаграммами. Устанавливается два соединения с 127.0.0.1 (localhost): `sock_1` и `sock_2`. С помощью `sock_2` отправляется сообщение на `sock_1`. С помощью `sock_1` получается сообщение. Отображается полученное сообщение. Оба соединения закрываются. Компьютеру так не слишком удобно взаимодействовать с самим собой, но пример показывает работу системы.

```
tarantool> socket = require('socket')
---
...
tarantool> sock_1 = socket('AF_INET', 'SOCK_DGRAM', 'udp')
---
...
tarantool> sock_1:bind('127.0.0.1')
---
- true
...
tarantool> sock_2 = socket('AF_INET', 'SOCK_DGRAM', 'udp')
---
...
tarantool> sock_2:sendto('127.0.0.1', sock_1:name().port,'X')
---
- 1
...
tarantool> message = sock_1:recvfrom(512)
---
...
tarantool> message
---
- X
...
tarantool> sock_1:close()
---
- true
...
tarantool> sock_2:close()
---
- true
...
```

### Использование tcp_server для получения содержимого файла, отправленного по socat

Ниже приведен пример функции `tcp_server`, которая читает строки с клиента и выводит результат. На клиентской стороне утилита `socat` в Linux будет использоваться для отправки целого файла на чтение функции `tcp_server`.

Запустите две оболочки. Первая оболочка будет экземпляром сервера. Вторая оболочка будет клиентом.

В первой оболочке запустите Tarantool и выполните:

```
box.cfg{}
socket = require('socket')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
socket.tcp_server('0.0.0.0', 3302,
{
  handler = function(s)
    while true do
      local request
      request = s:read("\n");
      if request == "" or request == nil then
        break
      end
      print(request)
    end
  end,
  prepare = function()
    print('Initialized')
  end
}
)
```

Вышеуказанный код означает:

1. Использовать `tcp_server()` для ожидания подключения с любого хоста по порту 3302.
2. Когда это произойдет, ввести цикл, который читает по сокету и выводит результат чтения. Разделителем для функции чтения будет «\n», поэтому каждое выполнение `read()` выполнит чтение строки до перевода строки, включая перевод строки.

Во второй оболочке создайте файл, который содержит несколько строк. Содержимое не имеет значения. Предположим, что первая строка содержит A, вторая строка содержит B, третья строка содержит C. Назовите этот файл «tmp.txt».

Во второй оболочке используйте утилиту socat для отправки файла tmp.txt на экземпляр сервера по хосту и порту:

```
$ socat TCP:localhost:3302 ./tmp.txt
```

Теперь смотрите, что происходит в первой оболочке. Выводятся строки «A», «B», «C».

### Использование tcp_server с handler и prepare

Ниже приведен пример функции tcp_server с использованием `handler` и `prepare`.

Запустите две оболочки. Первая оболочка будет экземпляром сервера. Вторая оболочка будет клиентом.

В первой оболочке запустите Tarantool и выполните:

```
box.cfg{}
socket = require('socket')
sock = socket.tcp_server(
  '0.0.0.0',
  3302,
  {prepare =
     function(sock)
       print('listening on socket ' .. sock:fd())
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      sock:setsockopt('SOL_SOCKET','SO_REUSEADDR',true)
      return 5
    end,
  handler =
   function(sock, from)
     print('accepted connection from: ')
     print('  host: ' .. from.host)
     print('  family: ' .. from.family)
     print('  port: ' .. from.port)
   end
 }
)
```

Вышеуказанный код означает:

1. Использовать `tcp_server()` для ожидания подключения с любого хоста по порту 3302.
2. Указать, что будет первый вызов `prepare`, который покажет что-то о сервере, затем вызовет `setsockopt(...'SO_REUSEADDR'...)` (это та же самая опция, которую Тарантул бы установил, если бы не было `prepare`), а затем вернет 5 (это довольно низкий размер очереди бэклога).
3. Указать, что будут вызовы `handler` по каждому соединению, которые будут отображать что-то о клиенте.

Теперь смотрите, что происходит в первой оболочке. Выведется что-то вроде „listening on socket 12“.

Во второй оболочке запустите Tarantool и выполните:

```
box.cfg{}
require('socket').tcp_connect('127.0.0.1', 3302)
```

Теперь смотрите, что происходит на первой оболочке. На дисплее появится что-то вроде „accepted connection from host: 127.0.0.1 family: AF_INET port: 37186“.

### 9.5.30 Модуль strict

Модуль `strict` включает в себя функции для включения или отключения строгого режима «strict mode». Когда включен строгий режим, попытка использовать необъявленную глобальную переменную приведет к ошибке. Глобальная переменная считается необъявленной, если ей никогда не было присвоено значение. Часто это указывает на ошибку программирования.

По умолчанию, строгий режим отключен, не считая случаев, когда сборка Tarantool производилась с помощью `-DCMAKE_BUILD_TYPE=Debug` – см. варианты сборки в разделе *сборка из исходников*.

**Пример:**

```
tarantool> strict = require('strict')
---
...
tarantool> strict.on()
---
...
tarantool> a = b -- строгий режим включен, поэтому появляется ошибка
---
- error: ... variable ''b'' is not declared'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> strict.off()
---
...
tarantool> a = b -- строгий режим отключен, поэтому ошибки нет
---
...
```

### 9.5.31 Модуль string

#### Общие сведения

Модуль `string` включает в себя всё из стандартной библиотеки для работы со строками в Lua, а также некоторые расширения специально для Tarantool.

В данном разделе мы рассматриваем только дополнительные функции, добавленные разработчиками Tarantool.

#### Указатель

Ниже приведен перечень всех функций библиотеки `string`.

| Имя | Назначение |
|---|---|
| *string.ljust()* | Выравнивание строки по левому полю |
| *string.rjust()* | Выравнивание строки по правому полю |
| *string.hex()* | Given a string, return hexadecimal values |
| *string.fromhex()* | Given hexadecimal values, return a string |
| *string.startswith()* | Проверка, начинается ли строка с заданной подстроки |
| *string.endswith()* | Проверка, заканчивается ли строка на заданную подстроку |
| *string.lstrip()* | Remove characters from the left of a string |
| *string.rstrip()* | Remove characters from the right of a string |
| *string.split()* | Разделение строки на таблицу со строками |
| *string.strip()* | Удаление пробелов слева и справа от строки |

`string.ljust`(*input-string*, *width*[, *pad-character* ])

Возврат строки, выровненной по левому краю, шириной, указанной в `width`.

**Параметры**

- `input-string` (*string*) – строка для выравнивания по левому краю
- `width` (`integer`) – ширина строки после выравнивания по левому краю
- `pad-character` (*string*) – отдельный символ, по умолчанию = 1 пробел

**Возвращается** выровненная по левому краю строка (не изменяется, если ширина <= длине строки)

**Тип возвращаемого значения** строка

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.ljust(' A', 5)
---
- ' A   '
...
```

string.rjust(*input-string*, *width*[, *pad-character*])

Возврат строки, выровненной по правому краю, шириной, указанной в `width`.

**Параметры**

- `input-string` (*string*) – строка для выравнивания по правому краю
- `width` (`integer`) – ширина строки после выравнивания по правому краю
- `pad-character` (*string*) – отдельный символ, по умолчанию = 1 пробел

**Возвращается** выровненная по правому краю строка (не изменяется, если ширина <= длине строки)

**Тип возвращаемого значения** строка

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.rjust('', 5, 'X')
---
- 'XXXXX'
...
```

string.hex(*input-string*)

Возврат шестнадцатеричного значения введенной строки.

**Параметры**

- `input-string` (*string*) – обрабатываемая строка

**Возвращается** шестнадцатеричное число, два символа шестнадцатеричных цифр для каждого введенного символа

**Тип возвращаемого значения** строка

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.hex('ABC ')
---
- '41424320'
...
```

string.fromhex(*hexadecimal-input-string*)

Given a string containing pairs of hexadecimal digits, return a string with one byte for each pair. This is the reverse of `string.hex()`. The hexadecimal-input-string must contain an even number of hexadecimal digits.

**Параметры**

- hexadecimal-input-string (*string*) – string with pairs of hexadecimal digits

**Возвращается** string with one byte for each pair of hexadecimal digits

**Тип возвращаемого значения** строка

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.fromhex('41424320')
---
- 'ABC '
...
```

string.startswith(*input-string*, *start-string*[, *start-pos*[, *end-pos*]])

Возврат true (правда), если input-string начинается со start-string, в противном случае, возврат false (ложь).

**Параметры**

- input-string (*string*) – строка, где производится поиск данных из start-string
- start-string (*string*) – искомая строка
- start-pos (integer) – положение: где начинать искать в пределах input-string
- end-pos (integer) – положение: где заканчивать искать в пределах input-string

**Возвращается** true (правда) или false (ложь)

**Тип возвращаемого значения** boolean (логический)

Значения start-pos и end-pos могут быть отрицательными, что означает, что положение вычисляется с конца строки.

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.startswith(' A', 'A', 2, 5)
---
- true
...
```

string.endswith(*input-string*, *end-string*[, *start-pos*[, *end-pos*]])

Возврат true (правда), если input-string заканчивается на end-string, в противном случае, возврат false (ложь).

**Параметры**

- input-string (*string*) – строка, где производится поиск данных из end-string
- end-string (*string*) – искомая строка
- start-pos (integer) – положение: где начинать искать в пределах input-string

- `end-pos` (`integer`) – положение: где заканчивать искать в пределах `input-string`

**Возвращается** true (правда) или false (ложь)

**Тип возвращаемого значения** boolean (логический)

Значения `start-pos` и `end-pos` могут быть отрицательными, что означает, что положение вычисляется с конца строки.

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.endswith('Baa', 'aa')
---
- true
...
```

`string.lstrip(`*input-string*`[`, *list-of-characters*`])`

Return the value of the input string, after removing characters on the left. The optional `list-of-characters` parameter is a set not a sequence, so `string.lstrip(...,'ABC')` does not mean strip `'ABC'`, it means strip `'A'` or `'B'` or `'C'`.

**Параметры**

- `input-string` (*string*) – обрабатываемая строка
- `list-of-characters` (*string*) – what characters can be stripped. Default = space.

**Возвращается** result after stripping characters from input string

**Тип возвращаемого значения** строка

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.lstrip(' ABC ')
---
- 'ABC '
...
```

`string.rstrip(`*input-string*`[`, *list-of-characters*`])`

Return the value of the input string, after removing characters on the right. The optional `list-of-characters` parameter is a set not a sequence, so `string.rstrip(...,'ABC')` does not mean strip `'ABC'`, it means strip `'A'` or `'B'` or `'C'`.

**Параметры**

- `input-string` (*string*) – обрабатываемая строка
- `list-of-characters` (*string*) – what characters can be stripped. Default = space.

**Возвращается** result after stripping characters from input string

**Тип возвращаемого значения** строка

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.rstrip(' ABC ')
---
- ' ABC'
...
```

string.split(*input-string*[, *split-string*[, *max* ]])

Разделение `input-string` на одну или более выводимых строк в таблице. Места разделения указаны в `split-string`.

**Параметры**

- `input-string` (*string*) – строка для разделения
- `split-string` (*string*) – искомая строка в пределах `input-string`. По умолчанию = пробел.
- `max` (`integer`) – максимальное количество символов-разделителей от начала обрабатываемой строки. Результат содержит не более max + 1 частей.

**Возвращается** таблица строк, которые были разделены из `input-string`

**Тип возвращаемого значения** таблица

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.split("A:B:C:D:F", ":", 2)
---
- - A
  - B
  - C:D:F
...
```

string.strip(*input-string*[, *list-of-characters* ])

Return the value of the input string, after removing characters on the left and the right. The optional `list-of-characters` parameter is a set not a sequence, so `string.strip(...,'ABC')` does not mean strip `'ABC'`, it means strip `'A'` or `'B'` or `'C'`.

**Параметры**

- `input-string` (*string*) – обрабатываемая строка
- `list-of-characters` (*string*) – what characters can be stripped. Default = space.

**Возвращается** result after stripping characters from input string

**Тип возвращаемого значения** строка

**Пример:**

```
tarantool> string = require('string')
---
...
tarantool> string.strip(' ABC ')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
- ABC
...
```

### 9.5.32 Модуль swim

#### Общие сведения

The `swim` module contains Tarantool's implementation of SWIM – Scalable Weakly-consistent Infection-style Process Group Membership Protocol. It is recommended for any type of Tarantool cluster where the number of nodes can be large. Its job is to discover and monitor the other members in the cluster and keep their information in a «member table». It works by sending and receiving, in a background event loop, periodically, via UDP, messages.

Each message has several parts, including:

- the ping such as «I am checking whether you are alive»,
- the event such as «I am joining»,
- the anti-entropy such as «I know that another member exists»,
- the payload such as «I or another member could have user-generated data».

The maximum message size is about 1500 bytes.

SWIM sends messages periodically to a random subset of the member table. SWIM processes replies from those members asynchronously.

Each entry in the member table has:

- UUID,
- a status («alive», «suspected», «dead», or «left»).

When a member fails to acknowledge a certain number of pings, its status is changed from «alive» to «suspected», that is, suspected of being dead. But SWIM tries to **avoid false positives** (misidentifying members as dead) which could happen when a member is overloaded and responds to pings too slowly, or when there is network trouble and packets can not go through some channels. When a member is suspected, SWIM randomly chooses other members and sends requests to them: «please ping this suspected member». This is called an **indirect ping**. Thus via different routes and additional hops the suspected member gets additional chances to reply, and thus «refute» the suspicion.

Because selection is random there is an **even network load** of about one message per member per protocol step, regardless of the cluster size. This is a major feature of SWIM. Because the protocol depends on members passing information on, also known as «gossiping», members do not need to broadcast messages to every member, which would cause a network load of N messages per member per protocol step, where N is the number of members in the cluster. However, selection is not entirely random, there is a preference for selecting least-recently-pinged members, like a round-robin.

Regarding the **anti-entropy** part of a message: this is necessary for maintaining the status in entries of the member table. Consider an example where two members, #1 and #2, are both alive. No events happen so only pings are being sent periodically. Then a third member, #3 appears. It knows about one of the existing members, #2. How can it discover the other member? Certainly #1 could notify #2 and #2 could notify #3, but messages go via UDP, so any notification event can be lost. However, regular messages containing «ping» and/or «event» also can contain an «anti-entropy» section, which is taken from a randomly-chosen part of the member table. So for this example, #2 will eventually randomly add to a regular message the

anti-entropy note that #1 is alive, and thus #3 will discover #1 even though it did not receive a direct «I am alive» event message from #1.

Regarding the **UUID** part of an entry in the member table: this is necessary for stable identification, because UUID changes more rarely than URI (a combination of IP and port number). But if the UUID does change, SWIM will include both the new and old UUID in messages, so all other members will eventually learn about the new UUID and change the member table accordingly.

Regarding the **payload** part of a message: this is not always necessary, it is a feature which allows passing user-generated information via SWIM instead of via node-to-node communication. The swim module has methods for specifying a «payload», which is arbitrary user data with a maximum size of about 1.2 KB. The payload can be anything, and it will be eventually disseminated over the cluster and available at other members. Each member can have its own payload.

Messages can be **encrypted**. Encryption may not be necessary in a closed network but is necessary for safety if the cluster is on the public Internet. Users can specify an encryption algorithm, an encryption mode, and a private key. All parts of all messages (including ping, acknowledgment, event, payload, URI, and UUID) will be encrypted with that private key, as well as a random public key generated for each message to prevent pattern attacks.

In theory the event dissemination speed (the number of hops to pass information throughout the cluster) is `O(log(cluster_size))`. For that and other theoretical information see the Cornell University paper which originally described SWIM.

`swim.new(`[*cfg*]`)`

Create a new SWIM instance. A SWIM instance maintains a member table and interacts with other members. Multiple SWIM instances can be created in a single Tarantool process.

**Параметры**

- `cfg` (*table*) – an optional configuration parameter.

  If `cfg` is not specified or is nil, then the new SWIM instance is not bound to a socket and has nil attributes, so it cannot interact with other members and only a few methods are valid until *swim_object:cfg()* is called.

  If `cfg` is specified, then the effect is the same as calling `s = swim.new() s:cfg()`, except for generation. For configuration description see *swim_object:cfg()*.

The generation part of `cfg` can only be specified during `new()`, it cannot be specified later during `cfg()`. Generation is part of *incarnation*. Usually generation is not specified because the default value (a timestamp) is sufficient, but if there is reason to mistrust timestamps (because the time is changed or because the instance is started on a different machine), then users may say `swim.new({generation = <number>})`. In that case the latest value should be persisted somehow (for example in a file, or in a space, or in a global service), and the new value must be greater than any previous value of generation.

**возвращает** swim-object *a swim object*

Пример:

```
swim_object = swim.new({uri = 3333, uuid = '00000000-0000-1000-8000-000000000001',␣
↪heartbeat_rate = 0.1})
```

`object swim_object`

A swim object is an object returned by *swim.new()*. It has methods: *cfg()*, *delete()*, *is_configured()*, *size()*, *quit()*, *add_member()*, *remove_member()*, *probe_member()*, *broadcast()*, *set_payload()*, *set_payload_raw()*, *set_codec()*, *self()*, *member_by_uuid()*, *pairs()*.

`swim_object:cfg(`*cfg*`)`

Configure or reconfigure a SWIM instance.

**Параметры**

- `cfg` (*table*) – the options to describe instance behavior

The `cfg` table may have these components:

- `heartbeat_rate` (double) – rate of sending round messages, in seconds. Setting `heartbeat_rate` to X does not mean that every member will be checked every X seconds, instead X is the protocol speed. Protocol period depends on member count and heartbeat_rate. Default = 1.
- `ack_timeout` (double) – time in seconds after which a ping is considered to be unacknowledged. Default = 30.
- `gc_mode` (enum) – dead member collection mode.

  If `gc_mode == 'off'` then SWIM never removes dead members from the member table (though users may remove them with *swim_object:remove_member()*), and SWIM will continue to ping them as if they were alive.

  If `gc_mode == 'on'` then SWIM removes dead members from the member table after one round.

  Default = `'on'`.
- `uri` (string or number) – either an `'ip:port'` address, or just a port number (if ip is omitted then 127.0.0.1 is assumed). If `port == 0`, then the kernel will select any free port for the IP address.
- `uuid` (string or cdata struct tt_uuid) – a value which should be unique among SWIM instances. Users may choose any value but the recommendation is: use *box.cfg.instance_uuid*, the Tarantool instance's UUID.

All the `cfg` components are dynamic – `swim_object:cfg()` may be called more than once. If it is not being called for the first time and a component is not specified, then the component retains its previous value. If it is being called for the first time then uri and uuid are mandatory, since a SWIM instance cannot operate without URI and UUID.

`swim_object:cfg()` is atomic – if there is an error, then nothing changes.

**возвращает** true if configuration succeeds

**возвращает** nil, `err` if an error occurred. `err` is an error object

Пример:

```
swim_object:cfg({heartbeat_rate = 0.5})
```

After `swim_object:cfg()`, all other `swim_object` methods are callable.

`.cfg`

Expose all non-nil components of the read-only table which was set up or changed by *swim_object:cfg()*.

Пример:

```
tarantool> swim_object.cfg
---
- gc_mode: off
  uri: 3333
  uuid: 00000000-0000-1000-8000-000000000001
...
```

`swim_object:delete()`
Delete a SWIM instance immediately. Its memory is freed, its member table entry is deleted, and it can no longer be used. Other members will treat this member as „dead“.

After `swim_object:delete()` any attempt to use the deleted instance will cause an exception to be thrown.

**возвращает** none, this method does not fail

Example: `swim_object:delete()`

`swim_object:is_configured()`
Return false if a SWIM instance was created via *swim.new()* without an optional `cfg` argument, and was not configured with *swim_object:cfg()*. Otherwise return true.

**возвращает** boolean result, true if configured, otherwise false

Example: `swim_object:is_configured()`

`swim_object:size()`
Return the size of the member table. It will be at least 1 because the «self» member is included.

**возвращает** integer size

Example: `swim_object:size()`

`swim_object:quit()`
Leave the cluster.

This is a graceful equivalent of *swim_object:delete()* – the instance is deleted, but before deletion it sends to each member in its member table a message, that this instance has left the cluster, and should not be considered dead.

Other instances will mark such a member in their tables as „left“, and drop it after one round of dissemination.

Consequences to the caller are the same as after `swim_object:delete()` – the instance is no longer usable, and an error will be thrown if there is an attempt to use it.

**возвращает** none, the method does not fail

Example: `swim_object:quit()`

`swim_object:add_member(`*cfg*`)`
Explicitly add a member into the member table.

This method is useful when a new member is joining the cluster and does not yet know what members already exist. In that case it can start interaction explicitly by adding the details about an already-existing member into its member table. Subsequently SWIM will discover other members automatically via messages from the already-existing member.

**Параметры**

- `cfg` (*table*) – description of the member

The `cfg` table has two mandatory components, `uuid` and `uri`, which have the same format as `uuid` and `uri` in the table for *swim_object:cfg()*.

**возвращает** true if member is added

**возвращает** nil, `err` if an error occurred. `err` is an error object

Пример:

```
swim_member_object = swim_object:add_member({uuid = ..., uri = ...})
```

`swim_object:remove_member(`*uuid*`)`
Explicitly and immediately remove a member from the member table.

**Параметры**

- `uuid` (`string-or-cdata-struct-tt_uuid`) – UUID

**возвращает** true if member is removed

**возвращает** nil, `err` if an error occurred. `err` is an error object.

Example: `swim_object:delete('00000000-0000-1000-8000-000000000001')`

`swim_object:probe_member(`*uri*`)`
Send a ping request to the specified `uri` address. If another member is listening at that address, it will receive the ping, and respond with an ACK (acknowledgment) message containing information such as UUID. That information will be added to the member table.

`swim_object:probe_member()` is similar to *swim_object:add_member()*, but it does not require UUID, and it is not reliable because it uses UDP.

**Параметры**

- `uri` (`string-or-number`) – URI. Format is the same as for `uri` in *swim_object:cfg()*.

**возвращает** true if member is pinged

**возвращает** nil, `err` if an error occurred. `err` is an error object.

Example: `swim_object:probe_member(3333)`

`swim_object:broadcast(`[*port*]`)`
Broadcast a ping request to all the network interfaces in the system.

`swim_object:broadcast()` is like *swim_object:probe_member()* to many members at once.

**Параметры**

- `port` (`number`) – All the sent ping requests have this port as destination port in their UDP headers. By default a currently bound port is used.

**возвращает** true if broadcast is sent

**возвращает** nil, `err` if an error occurred. `err` is an error object.

**Пример:**

```
tarantool> fiber = require('fiber')
---
...
tarantool> swim = require('swim')
---
...
tarantool> s1 = swim.new({uri = 3333, uuid = '00000000-0000-1000-8000-
↪000000000001', heartbeat_rate = 0.1})
---
...
tarantool> s2 = swim.new({uri = 3334, uuid = '00000000-0000-1000-8000-
↪000000000002', heartbeat_rate = 0.1})
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...
tarantool> s1:size()
---
- 1
...
tarantool> s1:add_member({uri = s2:self():uri(), uuid = s2:self():uuid()})
---
- true
...
tarantool> s1:size()
---
- 1
...
tarantool> s2:size()
---
- 1
...

tarantool> fiber.sleep(0.2)
---
...
tarantool> s1:size()
---
- 2
...
tarantool> s2:size()
---
- 2
...
tarantool> s1:remove_member(s2:self():uuid()) s2:remove_member(s1:self():uuid())
---
...
tarantool> s1:size()
---
- 1
...
tarantool> s2:size()
---
- 1
...

tarantool> s1:probe_member(s2:self():uri())
---
- true
...
tarantool> fiber.sleep(0.1)
---
...
tarantool> s1:size()
---
- 2
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> s2:size()
---
- 2
...
tarantool> s1:remove_member(s2:self():uuid()) s2:remove_member(s1:self():uuid())
---
...
tarantool> s1:size()
---
- 1
...
tarantool> s2:size()
---
- 1
...
tarantool> s1:broadcast(3334)
---
- true
...
tarantool> fiber.sleep(0.1)
---
...
tarantool> s1:size()
---
- 2
...

tarantool> s2:size()
---
- 2
...
```

`swim_object:set_payload(`*payload*`)`

Set a payload, as formatted data.

Payload is arbitrary user defined data up to 1200 bytes in size and disseminated over the cluster. So each cluster member will eventually learn what is the payload of other members in the cluster, because it is stored in the member table and can be queried with *swim_member_object:payload()*.

Different members may have different payloads.

**Параметры**

- `payload` (`object`) – Arbitrary Lua object to disseminate. Set to nil to remove the payload, in which case it will be eventually removed on other instances. The object is serialized in MessagePack.

**возвращает** true if payload is set

**возвращает** nil, `err` if an error occurred. `err` is an error object

Пример:

```
swim_object:set_payload({field1 = 100, field2 = 200})
```

swim_object:set_payload_raw(*payload*[, *size*])

Set a payload, as raw data.

Sometimes a payload does not need to be a Lua object. For example, a user may already have a well formatted MessagePack object and just wants to set it as a payload. Or cdata needs to be exposed.

set_payload_raw allows setting a payload as is, without MessagePack serialization.

**Параметры**

- payload (string-or-cdata) – any value
- size (number) – Payload size in bytes. If payload is string then size is optional, and if specified, then should not be larger than actual payload size. If size is less than actual payload size, then only the first size bytes of payload are used. If payload is cdata then size is mandatory.

**возвращает** true if payload is set

**возвращает** nil, err if an error occurred. err is an error object

Пример:

```
tarantool> tarantool> ffi = require('ffi')
---
...
tarantool> fiber = require('fiber')
---
...
tarantool> swim = require('swim')
---
...
tarantool> s1 = swim.new({uri = 0, uuid = '00000000-0000-1000-8000-000000000001
↪', heartbeat_rate = 0.1})
---
...
tarantool> s2 = swim.new({uri = 0, uuid = '00000000-0000-1000-8000-000000000002
↪', heartbeat_rate = 0.1})
---
...
tarantool> s1:add_member({uri = s2:self():uri(), uuid = s2:self():uuid()})
---
- true
...
tarantool> s1:set_payload({a = 100, b = 200})
---
- true
...
tarantool> s2:set_payload('any payload')
---
- true
...
tarantool> fiber.sleep(0.2)
---
...
tarantool> s1_view = s2:member_by_uuid(s1:self():uuid())
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
...
tarantool> s2_view = s1:member_by_uuid(s2:self():uuid())
---
...
tarantool> s1_view:payload()
---
- {'a': 100, 'b': 200}
...
tarantool> s2_view:payload()
---
- any payload
...
tarantool> cdata = ffi.new('char[?]', 2)
---
...
tarantool> cdata[0] = 1
---
...
tarantool> cdata[1] = 2
---
...
tarantool> s1:set_payload_raw(cdata, 2)
---
- true
...
tarantool> fiber.sleep(0.2)
---
...
tarantool> cdata, size = s1_view:payload_cdata()
---
...
tarantool> cdata[0]
---
- 1
...
tarantool> cdata[1]
---
- 2
...
tarantool> size
---
- 2
...
```

`swim_object:set_codec(`*codec_cfg*`)`

Enable encryption for all following messages.

For a brief description of encryption algorithms see «enum_crypto_algo» and «enum crypto_mode» in the Tarantool source code file crypto.h.

When encryption is enabled, all the messages are encrypted with a chosen private key, and a randomly generated and updated public key.

**Параметры**

- `codec_cfg` (*table*) – description of the encryption

The components of the `codec_cfg` table may be:

- `algo` (string) – encryption algorithm name. All the names in *module crypto* are supported: „aes128“, „aes192“, „aes256“, „des“. Specify „none“ to disable encryption.
- `mode` (string) – encryption algorithm mode. All the modes in module `crypto` are supported: „ecb“, „cbc“, „cfb“, „ofb“. Default = „cbc“.
- `key` (cdata or string) – a private secret key which is kept secret and should never be stored hard-coded in source code.
- `key_size` (integer) – size of the key in bytes.

  `key_size` is mandatory if key is cdata.

  `key_size` is optional if key is string, and if `key_size` is shorter than than actual key size then the key is truncated.

All of `algo`, `mode`, `key`, and `key_size` should be the same for all SWIM instances, so that members can understand each others“ messages.

Пример;

```
tarantool> tarantool> swim = require('swim')
---
...
tarantool> s1 = swim.new({uri = 0, uuid = '00000000-0000-1000-8000-000000000001
↪'})
---
...
tarantool> s1:set_codec({algo = 'aes128', mode = 'cbc', key = '1234567812345678
↪'})
---
- true
...
```

`swim_object:self()`

Return a *swim member object* (of self) from the member table, or from a cache containing earlier results of `swim_object:self()` or `swim_object:member_by_uuid()` or `swim_object:pairs()`.

**возвращает** *swim member object*, not nil because self() will not fail

Example: `swim_member_object = swim_object:self()`

`swim_object:member_by_uuid(`*uuid*`)`

Return a *swim member object* (given UUID) from the member table, or from a cache containing earlier results of `swim_object:self()` or `swim_object:member_by_uuid()` or `swim_object:pairs()`.

**Параметры**

- `uuid` (`string-or-cdata-struct-tt-uuid`) – UUID

**возвращает** *swim member object*, or nil if not found

Пример:

```
swim_member_object = swim_object:member_by_uuid('00000000-0000-1000-8000-
↪000000000001')
```

(continues on next page)

(продолжение с предыдущей страницы)

`swim_object:pairs()`

Set up an iterator for returning *swim member objects* from the member table, or from a cache containing earlier results of `swim_object:self()` or `swim_object:member_by_uuid()` or `swim_object:pairs()`.

`swim_object:pairs()` should be in a „for“ loop, and there should only be one iterator in operation at one time. (The iterator is implemented in an extra light fashion so only one iterator object is available per SWIM instance.)

**Параметры**

- `generator+object+key` (`varies`) – as for any Lua pairs() iterators. generator function, iterator object (a swim member object), and initial key (a UUID).

Пример:

```
tarantool> fiber = require('fiber')
---
...
tarantool> swim = require('swim')
---
...
tarantool> s1 = swim.new({uri = 0, uuid = '00000000-0000-1000-8000-000000000001
↪', heartbeat_rate = 0.1})
---
...
tarantool> s2 = swim.new({uri = 0, uuid = '00000000-0000-1000-8000-000000000002
↪', heartbeat_rate = 0.1})
---
...
tarantool> s1:add_member({uri = s2:self():uri(), uuid = s2:self():uuid()})
---
- true
...
tarantool> fiber.sleep(0.2)
---
...
tarantool> s1:self()
---
- uri: 127.0.0.1:55845
  status: alive
  incarnation: cdata {generation = 1569353431853325ULL, version = 1ULL}
  uuid: 00000000-0000-1000-8000-000000000001
  payload_size: 0
...
tarantool> s1:member_by_uuid(s1:self():uuid())
---
- uri: 127.0.0.1:55845
  status: alive
  incarnation: cdata {generation = 1569353431853325ULL, version = 1ULL}
  uuid: 00000000-0000-1000-8000-000000000001
  payload_size: 0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> s1:member_by_uuid(s2:self():uuid())
---
- uri: 127.0.0.1:53666
  status: alive
  incarnation: cdata {generation = 1569353431865138ULL, version = 1ULL}
  uuid: 00000000-0000-1000-8000-000000000002
  payload_size: 0
...
tarantool> t = {}
---
...
tarantool> for k, v in s1:pairs() do table.insert(t, {k, v}) end
---
...
tarantool> t
---
- - - 00000000-0000-1000-8000-000000000002
    - uri: 127.0.0.1:53666
      status: alive
      incarnation: cdata {generation = 1569353431865138ULL, version = 1ULL}
      uuid: 00000000-0000-1000-8000-000000000002
      payload_size: 0
  - - 00000000-0000-1000-8000-000000000001
    - uri: 127.0.0.1:55845
      status: alive
      incarnation: cdata {generation = 1569353431853325ULL, version = 1ULL}
      uuid: 00000000-0000-1000-8000-000000000001
      payload_size: 0
...
```

`object swim_member_object`

Methods *swim_object:member_by_uuid()*, *swim_object:self()*, and *swim_object:pairs()* return swim member objects.

A swim member object has methods for reading its attributes: *status()*, *uuid*, *uri()*, *incarnation()*, *payload_cdata*, *payload_str()*, *payload()*, *is_dropped()*.

`swim_member_object:status()`

Return the status, which may be „alive“, „suspected“, „left“, or „dead“.

**возвращает** string „alive“ | „suspected“ | „left“ | dead“

`swim_member_object:uuid()`

Return the UUID as cdata struct tt_uuid.

**возвращает** cdata-struct-tt-uuid UUID

`swim_member_object:uri()`

Return the URI as a string „ip:port“. Via this method a user can learn a real assigned port, if port = 0 was specified in *swim_object:cfg()*.

**возвращает** string ip:port

`swim_member_object:incarnation()`

Return a cdata object with the *incarnation*. The cdata object has two attributes: incarnation().generation and incarnation().version.

Incarnations can be compared to each other with any comparison operator (==, <, >, <=, >=, ~=).

Incarnations, when printed, will appear as strings with both generation and version.

**возвращает** cdata incarnation

`swim_member_object:payload_cdata()`
Return member's payload.

**возвращает** pointer-to-cdata payload and size in bytes

`swim_member_object:payload_str()`
Return payload as a string object. Payload is not decoded. It is just returned as a string instead of cdata. If payload was not specified by *swim_object:set_payload()* or by *swim_object:set_payload_raw()*, then its size is 0 and nil is returned.

**возвращает** string-object payload, or nil if there is no payload

`swim_member_object:payload()`
Since the `swim` module is a Lua module, a user is likely to use Lua objects as a payload – tables, numbers, strings etc. And it is natural to expect that *swim_member_object:payload()* should return the same object which was passed into *swim_object:set_payload()* by another instance. `swim_member_object:payload()` tries to interpret payload as MessagePack, and if that fails then it returns the payload as a string.

`swim_member_object:payload()` caches its result. Therefore only the first call actually decodes cdata payload. All following calls return a pointer to the same result, unless payload is changed with a new incarnation. If payload was not specified (its size is 0), then nil is returned.

`swim_member_object:is_dropped()`
Returns true if this member object is a stray reference to a member which has already been dropped from the member table.

**возвращает** boolean true if member is dropped, otherwise false

Пример:

```
tarantool> swim = require('swim')
---
...
tarantool> s = swim.new({uri = 0, uuid = '00000000-0000-1000-8000-000000000001'}
↪)
---
...
tarantool> self = s:self()
---
...
tarantool> self:status()
---
- alive
...
tarantool> self:uuid()
---
- 00000000-0000-1000-8000-000000000001
...
tarantool> self:uri()
---
- 127.0.0.1:56367
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> self:incarnation()
---
- - cdata {generation = 1569354463981551ULL, version = 1ULL}
...
tarantool> self:is_dropped()
---
- false
...
tarantool> s:set_payload_raw('123')
---
- true
...
tarantool> self:payload_cdata()
---
- 'cdata<const char *>: 0x0103500050'
- 3
...
tarantool> self:payload_str()
---
- '123'
...
tarantool> s:set_payload({a = 100})
---
- true
...
tarantool> self:payload_cdata()
---
- 'cdata<const char *>: 0x0103500050'
- 4
...
tarantool> self:payload_str()
---
- !!binary gaFhZA==
...
tarantool> self:payload()
---
- {'a': 100}
...
```

`swim_member_object:on_member_event(`*trigger-function*`[, `*ctx*`])`

Create an «on_member *trigger*». The `trigger-function` will be executed when a member in the member table is updated.

**Параметры**

- `trigger-function` (`function`) – this will become the trigger function
- `ctx` (`cdata`) – (optional) this will be passed to trigger-function

**возвращает** nil or function pointer.

The **trigger-function** should have three parameter declarations (Tarantool will pass values for them when it invokes the function):

- the member which is having the member event,

- the event object,
- the `ctx` which will be the same value as what is passed to `swim_object:on_member_event`.

A **member event** is any of:

- appearance of a new member,
- drop of an existing member, or
- update of an existing member.

An **event object** is an object which the trigger-function can use for determining what type of member event has happened. The object's methods – such as `is_new_status()`, `is_new_uri()`, `is_new_incarnation()`, `is_new_payload()`, `is_drop()` – return boolean values.

A member event may have more than one associated **trigger**. Triggers are executed sequentially. Therefore if a trigger function causes yields or sleeps, other triggers may be forced to wait. However, since trigger execution is done in a separate fiber, SWIM itself is not forced to wait.

Example of an on-member trigger function:

```
tarantool> swim = require('swim')

local function on_event(member, event, ctx)
    if event:is_new() then
        ...
    elseif event:is_drop() then
        ...
    end

    if event:is_update() then
        -- All next conditions can be
        -- true simultaneously.
        if event:is_new_status() then
...
        end
        if event:is_new_uri() then
...
        end
        if event:is_new_incarnation() then
...
        end
        if event:is_new_payload() then
...
        end
    end
end
```

Notice in the above example that the function is ready for the possibility that multiple events can happen simultaneously for a single trigger activation. `is_new()` and `is_drop()` can not both be true, but `is_new()` and `is_update()` can both be true, or `is_drop()` and `is_update()` can both be true. Multiple simultaneous events are especially likely if there are many events and trigger functions are slow – in that case, for example, a member might be added and then updated after a while, and then after a while there will be a single trigger activation.

Also: `is_new()` and `is_new_payload()` can both be true. This case is not due to trigger functions that are slow. It occurs because «omitted payload» and «size-zero payload» are not the same thing. For example: when a ping is received, a new member might be added, but ping messages do not

include payload. The payload will appear later in a different message. If that is important for the application, then the function should not assume when `is_new()` is true that the member already has a payload, and should not assume that payload size says something about the payload's presence or absence.

Also: functions should not assume that `is_new()` and `is_drop()` will always be seen. If a new member appears but then is dropped before its appearance has caused a trigger activation, then there will be no trigger activation.

`is_new_generation()` will be true if the generation part of *incarnation* changes. `is_new_version()` will be true if the version part of incarnation changes. `is_new_incarnation()` will be true if either the generation part or the version part of incarnation changes. For example a combination of these methods can be used within a user-defined trigger to check whether a process has restarted, or a member has changed ...

```
swim = require('swim')
s = swim.new()
s:on_member_event(function(m, e)
...
    if e:is_new_incarnation() then
        if e:is_new_generation() then
            -- Process restart.
        end
        if e:is_new_version() then
            -- Process version update. It means
            -- the member is somehow changed.
        end
    end
end
```

`swim_member_object:on_member_event(`*nil*, *old-trigger*`)`
Delete an on-member trigger.

**Параметры**

- `old-trigger` (`function`) – old-trigger

The old-trigger value should be the value returned by `on_member_event(trigger-function[, ctx])`.

`swim_member_object:on_member_event(`*new-trigger*, *old-trigger*`[`, *ctx* `])`
This is a variation of `on_member_event(new-trigger, [, ctx])`.

The additional parameter is `old-trigger`. Instead of adding the new-trigger at the end of a list of triggers, this function will replace the entry in the list of triggers that matches old-trigger. The position within a list may be important because triggers are activated sequentially starting with the first trigger in the list.

The old-trigger value should be the value returned by `on_member_event(trigger-function[, ctx])`.

`swim_member_object:on_member_event()`
Return the list of on-member triggers.

### SWIM internals

The SWIM internals section is not necessary for programmers who wish to use the SWIM module, it is for programmers who wish to change or replace the SWIM module.

The SWIM wire protocol is open, will be backward compatible in case of any changes, and can be implemented by users who wish to simulate their own SWIM cluster members because they use another language than Lua, or another environment unrelated to Tarantool. The protocol is encoded as MsgPack.

```
SWIM packet structure:

+-----------------Public data, not encrypted-----------------+
|                                                            |
|      Initial vector, size depends on chosen algorithm.     |
|                  Next data is encrypted.                   |
|                                                            |
+----------Meta section, handled by transport level----------+
| map {                                                      |
|     0 = SWIM_META_TARANTOOL_VERSION: uint, Tarantool       |
|                                      version ID,           |
|     1 = SWIM_META_SRC_ADDRESS: uint, ip,                   |
|     2 = SWIM_META_SRC_PORT: uint, port,                    |
|     3 = SWIM_META_ROUTING: map {                           |
|         0 = SWIM_ROUTE_SRC_ADDRESS: uint, ip,              |
|         1 = SWIM_ROUTE_SRC_PORT: uint, port,               |
|         2 = SWIM_ROUTE_DST_ADDRESS: uint, ip,              |
|         3 = SWIM_ROUTE_DST_PORT: uint, port                |
|     }                                                      |
| }                                                          |
+-------------------Protocol logic section-------------------+
| map {                                                      |
|     0 = SWIM_SRC_UUID: 16 byte UUID,                       |
|                                                            |
|                 AND                                        |
|                                                            |
|     2 = SWIM_FAILURE_DETECTION: map {                      |
|         0 = SWIM_FD_MSG_TYPE: uint, enum swim_fd_msg_type, |
|         1 = SWIM_FD_GENERATION: uint,                      |
|         2 = SWIM_FD_VERSION: uint                          |
|     },                                                     |
|                                                            |
|               OR/AND                                       |
|                                                            |
|     3 = SWIM_DISSEMINATION: array [                        |
|         map {                                              |
|             0 = SWIM_MEMBER_STATUS: uint,                  |
|                                     enum member_status,    |
|             1 = SWIM_MEMBER_ADDRESS: uint, ip,             |
|             2 = SWIM_MEMBER_PORT: uint, port,              |
|             3 = SWIM_MEMBER_UUID: 16 byte UUID,            |
|             4 = SWIM_MEMBER_GENERATION: uint,              |
|             5 = SWIM_MEMBER_VERSION: uint,                 |
|             6 = SWIM_MEMBER_PAYLOAD: bin                   |
|         },                                                 |
```

(continues on next page)

(продолжение с предыдущей страницы)

```
|         ...                                                    |
|     ],                                                         |
|                                                                |
|                OR/AND                                          |
|                                                                |
|     1 = SWIM_ANTI_ENTROPY: array [                             |
|         map {                                                  |
|             0 = SWIM_MEMBER_STATUS: uint,                      |
|                                     enum member_status,        |
|             1 = SWIM_MEMBER_ADDRESS: uint, ip,                 |
|             2 = SWIM_MEMBER_PORT: uint, port,                  |
|             3 = SWIM_MEMBER_UUID: 16 byte UUID,                |
|             4 = SWIM_MEMBER_GENERATION: uint,                  |
|             5 = SWIM_MEMBER_VERSION: uint,                     |
|             6 = SWIM_MEMBER_PAYLOAD: bin                       |
|         },                                                     |
|         ...                                                    |
|     ],                                                         |
|                                                                |
|                OR/AND                                          |
|                                                                |
|     4 = SWIM_QUIT: map {                                       |
|         0 = SWIM_QUIT_GENERATION: uint,                        |
|         1 = SWIM_QUIT_VERSION: uint                            |
|     }                                                          |
| }                                                              |
+----------------------------------------------------------------+
```

The **Initial vector section** appears only when encryption is enabled. This section contains a public key. For example, for AES algorithms it is a 16-byte initial vector stored as is. When no encryption is used, the section size is 0.

The later sections (Meta and Protocol Logic) are encrypted as one big data chunk if encryption is enabled.

The **Meta section** handles routing and protocol versions compatibility. It works at the „transport“ level of the SWIM protocol, and is always present. Keys in the meta section are:

- SWIM_META_TARANTOOL_VERSION – mandatory field. Tarantool sets here its version as a 3 byte integer:
  - 1 byte for major,
  - 1 byte for minor,
  - 1 byte for patch.

  For example, Tarantool version 2.1.3 would be encoded like this: `(((2 << 8) | 1) << 8) | 3;`. This field will be used to support multiple versions of the protocol.
- SWIM_META_SRC_ADDRESS and SWIM_META_SRC_PORT – mandatory. source IP address and port. IP is encoded as 4 bytes. «xxx.xxx.xxx.xxx» where each „xxx“ is encoding of one byte. Port is encoded as an integer. Example of how to encode «127.0.0.1:3313»:

```
struct in_addr addr;
inet_aton("127.0.0.1", &addr);
pos = mp_encode_uint(pos, SWIM_META_SRC_ADDRESS);
```

(continues on next page)

(продолжение с предыдущей страницы)

```
pos = mp_encode_uint(pos, addr->s_addr);
pos = mp_encode_uint(pos, SWIM_META_SRC_PORT);
pos = mp_encode_uint(pos, 3313);
```

- SWIM_META_ROUTING subsection – not mandatory. Responsible for packet forwarding. Used by SWIM suspicion mechanism. Read about suspicion in the SWIM paper.

  If this subsection is present then the following fields are mandatory:

  - SWIM_ROUTE_SRC_ADDRESS and SWIM_ROUTE_SRC_PORT (source IP address and port) (should be an address of the message originator (can differ from
  - SWIM_META_SRC_ADDRESS and from SWIM_META_SRC_ADDRESS_PORT);
  - SWIM_ROUTE_DST_ADDRESS and SWIM_ROUTE_DST_PORT (destination IP address and port, for the message's final destination).

  If a message was sent indirectly with the help of SWIM_META_ROUTING, then the reply should be sent back by the same route.

  For an example of how SWIM uses routing for indirect pings ... Assume there are 3 nodes: S1, S2, S3. S1 sends a message to S3 via S2. The following steps are executed in order to deliver the message:

  ```
  S1 -> S2
  { src: S1, routing: {src: S1, dst: S3}, body: ... }
  ```

  S2 receives the message and sees that routing.dst is not equal to S2, so it is a foreign packet. S2 forwards the packet to S3 preserving all the data including body and routing sections.

  ```
  S2 -> S3
  ```

  S3 receives the message and sees that routing.dst is equal to S3, so the message is delivered. If S3 wants to answer, it sends a response via the same proxy. It knows that the message was delivered from S2, so it sends an answer via S2.

The **Protocol logic section** handles SWIM logical protocol steps and actions.

- SWIM_SRC_UUID – mandatory field. SWIM uses UUID as a unique identifier of a member, not IP/port. This field stores UUID of sender. Its type is MP_BIN. Size is always 16 bytes. UUID is encoded in host byte order, no bswaps are needed.

Following SWIM_SRC_UUID there are four possible subsections: SWIM_FAILURE_DETECTION, SWIM_DISSEMINATION, SWIM_ANTI_ENTROPY, SWIM_QUIT. Any or all of these subsections may be present. A connector should be ready to handle any combination.

- SWIM_FAILURE_DETECTION subsection – describes a ping or ACK. In the SWIM_FAILURE_DETECTION subsection are:
  - SWIM_FD_MSG_TYPE (0 is ping, 1 is ack);
  - SWIM_FD_GENERATION + SWIM_FD_VERSION (the *incarnation*).
- SWIM_DISSEMINATION subsection – a list of changed cluster members. It may include only a subset of changed cluster members if there are too many changes to fit into one UDP packet.

  In the SWIM_DISSEMINATION subsection are:

  - SWIM_MEMBER_STATUS (mandatory) (0 = alive, 1 = suspected, 2 = dead, 3 = left);
  - SWIM_MEMBER_ADDRESS and SWIM_MEMBER_PORT (mandatory) member IP and port;

  - SWIM_MEMBER_UUID (mandatory) (member UUID);
  - SWIM_MEMBER_GENERATION + SWIM_MEMBER_VERSION (mandatory) (the member *incarnation*);
  - SWIM_MEMBER_PAYLOAD (not mandatory) (member payload) (MessagePack type is MP_BIN).

  Note that absence of SWIM_MEMBER_PAYLOAD means nothing - it is not the same as a payload with zero size.

- SWIM_ANTI_ENTROPY subsection – a helper for the dissemination. It contains all the same fields as the dissemination sub, but all of them are mandatory, including payload even when payload size is 0. Anti-entropy eventually spreads changes which for any reason are not spread by the dissemination.
- SWIM_QUIT subsection – statement that the sender has left the cluster gracefully, for example via *swim_object:quit()*, and should not be considered dead. Sender status should be changed to „left“.

  In the SWIM_QUIT subsection are:

  - SWIM_QUIT_GENERATION + SWIM_QUIT_VERSION (the sender *incarnation*).

The **incarnation** is a 128-bit cdata value which is part of each member's configuration and is present in most messages. It has two parts: generation and version.

Generation is persistent. By default it has the number of microseconds since the epoch (compare the value returned by *clock_realtime64()*). Optionally a user can set generation during *new()*.

Version is volatile. It is initially 0. It is incremented automatically every time that a change occurs.

The incarnation, or sometimes the version alone, is useful for deciding to ignore obsolete messages, for updating a member's attributes on remote nodes, and for refuting messages that say a member is dead.

If the member's incarnation is less than the locally stored incarnation, then the message is obsolete. This can happen because UDP allows reordering and duplication.

If the member's incarnation in a message is greater than the locally stored incarnation, then most of its attributes (IP, port, status) should be updated with the values received in the message. However, the payload attribute should not be updated unless it is present in the message. Because of its relatively large size, payload is not always included in every message.

Refutation usually happens when a false-positive failure detection has happened. In such a case the member thought to be dead receives that information from other members, increases its own incarnation, and spreads a message saying the member is alive (a «refutation»).

Note: in the original version of Tarantool SWIM, and in the original SWIM specification, there is no generation and the incarnation consists of only the version. Generation was added because it is useful for detecting obsolete messages left over from a previous life of an instance that has restarted.

### 9.5.33 Модуль table

Модуль `table` включает в себя всё из стандартной библиотеки для работы с таблицами в Lua, а также некоторые расширения специально для Tarantool.

Write `table` to see the list of functions:

- `clear` (LuaJIT extension = erase all elements)
- concat (concatenate)
- `copy` (make a copy of an array)
- `deepcopy` (see the *description* below)

- `foreach`
- `foreachi`
- getn (get the number of elements in an array)
- insert (insert an element into an array)
- maxn (get the largest index)
- move (move elements between tables)
- `new` (LuaJIT extension = return a new table with pre-allocated elements)
- remove (remove an element from an array)
- sort (sort the elements of an array)

В данном разделе мы рассматриваем только дополнительную функцию, добавленную разработчиками Tarantool: `deepcopy`.

`table.deepcopy`(*input-table*)

Возврат детальной копии таблицы – копии, которая включает в себя вложенные структуры любой глубины и не зависит от указателей, копируется содержимое.

**Параметры**

- `input-table` – таблица для копирования

**Возвращается** копия таблицы

**Тип возвращаемого значения** таблица

**Пример:**

```
tarantool> input_table = {1,{'a','b'}}
---
...

tarantool> output_table = table.deepcopy(input_table)
---
...

tarantool> output_table
---
- - 1
  - - a
    - b
...
```

`table.sort`(*input-table*[, *comparison-function*])

Размещение содержимого введенной таблицы в отсортированном порядке.

В базовой сортировке в Lua, table.sort, есть функция сравнения, которая используется по умолчанию: `function (a, b) return a < b end`.

Эта стандартная функция эффективна. Однако иногда пользователям Tarantool может понадобиться эквивалент `table.sort` со следующими функциями:

1. If the table contains nils, except nils at the end, the results must still be correct. That is not the case with the default `tarantool_sort`, and it cannot be fixed by making a comparison that checks whether `a` and `b` are `nil`. (Before trying certain Internet suggestions, test with `{1, nil, 2, -1, 44, 1e308, nil, 2, nil, nil, 0}`.

2. If strings are to be sorted in a language-aware way, there must be a parameter for collation.
3. If the table has a mix of types, then they must be sorted as booleans, then numbers, then strings, then byte arrays.

Поскольку все эти функции доступны в спейсах Tarantool, решение простое: создайте временный спейс в Tarantool, поместите в него содержимое таблицы, извлеките из него кортежи по порядку и перезапишите таблицу.

Тогда `tarantool_sort()` сделает то же самое, что и `table.sort`, но с этими дополнительными функциями. Это не быстрый способ, который требует прав на базу данных, поэтому его следует использовать только при необходимости дополнительных функций.

**Пример:**

```
function tarantool_sort(input_table, collation)
    local c = collation or 'binary'
    local tmp_name = 'Temporary_for_tarantool_sort'
    pcall(function() box.space[tmp_name]:drop() end)
    box.schema.space.create(tmp_name, {temporary = true})
    box.space[tmp_name]:create_index('I')
    box.space[tmp_name]:create_index('I2',
                                     {unique = false,
                                      type='tree',
                                      parts={{2, 'scalar',
                                              collation = c,
                                              is_nullable = true}}})
    for i = 1, table.maxn(input_table) do
        box.space[tmp_name]:insert{i, input_table[i]}
    end
    local t = box.space[tmp_name].index.I2:select()
    for i = 1, table.maxn(input_table) do
        input_table[i] = t[i][2]
    end
    box.space[tmp_name]:drop()
  end
```

For example, suppose `table t = {1, 'A', -88.3, nil, true, 'b', 'B', nil, 'À'}`.

After `tarantool_sort(t, 'unicode_ci')` `t` contains `{nil, nil, true, -88.3, 1, 'A', 'À', 'b', 'B'}`.

### 9.5.34 Модуль tap

#### Общие сведения

The `tap` module streamlines the testing of other modules. It allows writing of tests in the TAP protocol. The results from the tests can be parsed by standard TAP-analyzers so they can be passed to utilities such as prove. Thus, one can run tests and then use the results for statistics, decision-making, and so on.

### Справочник по API

| Имя | Назначение |
|---|---|
| *tap.test()* | Инициализация |
| *taptest:test()* | Создание подтеста и вывод результатов |
| *taptest:plan()* | Указание количества проводимых тестов |
| *taptest:check()* | Проверка количества выполненных тестов |
| *taptest:diag()* | Отображение сообщения диагностики |
| *taptest:ok()* | Оценка состояния и отображение сообщения |
| *taptest:fail()* | Оценка состояния и отображение сообщения |
| *taptest:skip()* | Оценка состояния и отображение сообщения |
| *taptest:is()* | Проверка равенства двух аргументов |
| *taptest:isnt()* | Проверка отличий двух аргументов |
| *taptest:is_deeply()* | Рекурсивная проверка равенства двух аргументов |
| *taptest:like()* | Проверка соответствия аргумента шаблону |
| *taptest:unlike()* | Проверка отличия аргумента от шаблона |
| *taptest:isnil()*<br>*taptest:isstring()*<br>*taptest:isnumber()*<br>*taptest:istable()*<br>*taptest:isboolean()*<br>*taptest:isudata()*<br>*taptest:iscdata()* | Проверка соответствия значения определенному типу |
| *taptest.strict* | Flag, true if comparisons with `nil` should be strict |

`tap.test`(*test-name*)
Инициализация.

Результатом `tap.test` является объект, который будет называться taptest в ходе данного разбора, что необходимо для `taptest:plan()` и всех остальных методов.

**Параметры**

- `test-name` (*string*) – произвольное имя для результата теста.

**возвращает** taptest

**тип возвращаемого значения** таблица

```
tap = require('tap')
taptest = tap.test('test-name')
```

`object taptest`

`taptest:test`(*test-name*, *func*)
Создание подтеста (если не указан аргумент `func`) или (если указаны все аргументы) создание подтеста, выполнение тестовой функции и вывод результата.

См. *пример*.

**Параметры**

- `name` (*string*) – произвольное имя для результата теста.
- `fun` (`function`) – выполняемая тестовая логика.

**возвращает** taptest

**тип возвращаемого значения** userdata или строка

`taptest:plan(`*count*`)`
Указание количества проводимых тестов.

**Параметры**

- `count` (`number`) –

**возвращает** nil

`taptest:check()`
Проверка количества выполненных тестов.

Выведенный результат будет включать в себя сообщение: `# bad plan: ...`, если количество выполненных тестов не равно количеству тестов, указанному в `taptest:plan(...)`. (Это собственная функция Tarantool: сообщения типа «bad plan» не входят в стандарт TAP13.)

Такую проверку следует проводить только по завершении всех запланированных тестов, поэтому как правило, `taptest:check()` появится лишь в конце скрипта. Тем не менее, в качестве расширения Tarantool, `taptest:check()` может появиться в начале любого подтеста. Таким образом, проверка появится в трех случаях:

- при вызове `taptest:check()` в конце скрипта,
- при вызове функции, которая заканчивается вызовом `taptest:check()`,
- или при вызове taptest:test(„...“, имя-функции-подтеста), где функция подтеста не обязана заканчиваться на `taptest:check()`, поскольку ее можно вызвать по окончании подтеста.

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

`taptest:diag(`*message*`)`
Отображение сообщения диагностики.

**Параметры**

- `message` (*string*) – отображаемое сообщение.

**возвращает** nil

`taptest:ok(`*condition*, *test-name*`)`
Это базовая функция, которая используется другими функциями. В зависимости от условия `condition`, выводится „ok“ или „not ok“ вместе с отладочной информацией. Отображается сообщение.

**Параметры**

- `condition` (`boolean`) – выражение, которое либо true (правда), либо false (ложь)
- `test-name` (*string*) – имя теста

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

**Пример:**

```
tarantool> taptest:ok(true, 'x')
ok - x
---
- true
...
tarantool> tap = require('tap')
---
...
tarantool> taptest = tap.test('test-name')
TAP version 13
---
...
tarantool> taptest:ok(1 + 1 == 2, 'X')
ok - X
---
- true
...
```

taptest:fail(*test-name*)

taptest:fail('x') – аналог taptest:ok(false, 'x'). Отображается сообщение.

**Параметры**

- test-name (*string*) – имя теста

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

taptest:skip(*message*)

taptest:skip('x') – аналог taptest:ok(true, 'x' .. '# skip'). Отображается сообщение.

**Параметры**

- test-name (*string*) – имя теста

**возвращает** nil

**Пример:**

```
tarantool> taptest:skip('message')
ok - message # skip
---
- true
...
```

taptest:is(*got*, *expected*, *test-name*)

Проверка равенства первого аргумента второму аргументу. Отображается подробное сообщение, если результатом будет false (ложь).

**Параметры**

- got (number) – фактический результат
- expected (number) – ожидаемый результат
- test-name (*string*) – имя теста

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

taptest:isnt(*got*, *expected*, *test-name*)
Отрицание *taptest:is()*.

**Параметры**

- got (number) – фактический результат
- expected (number) – ожидаемый результат
- test-name (*string*) – имя теста

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

taptest:is_deeply(*got*, *expected*, *test-name*)
Recursive version of taptest:is(...), which can be used to compare tables as well as scalar values.

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

**Параметры**

- got (lua-value) – фактический результат
- expected (lua-value) – ожидаемый результат
- test-name (*string*) – имя теста

taptest:like(*got*, *expected*, *test-name*)
Проверка совпадения строки с шаблоном. Ok, если найдено совпадение.

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

**Параметры**

- got (lua-value) – фактический результат
- expected (lua-value) – шаблон
- test-name (*string*) – имя теста

```
test:like(tarantool.version, '^[1-9]', "version")
```

taptest:unlike(*got*, *expected*, *test-name*)
Отрицание *taptest:like()*.

**Параметры**

- got (number) – фактический результат
- expected (number) – шаблон
- test-name (*string*) – имя теста

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

taptest:isnil(*value*, *message*, *extra*)
taptest:isstring(*value*, *message*, *extra*)
taptest:isnumber(*value*, *message*, *extra*)
taptest:istable(*value*, *message*, *extra*)
taptest:isboolean(*value*, *message*, *extra*)
taptest:isudata(*value*, *utype*, *message*, *extra*)

taptest:iscdata(*value*, *ctype*, *message*, *extra*)
Проверка соответствия значения определенному типу. Отображается длинное сообщение, если значение не принадлежит указанному типу.

**Параметры**

- `value` (`lua-value`) – value the type of which is to be checked
- `utype` (*string*) – type of data that a passed value should have
- `ctype` (*string*) – type of data that a passed value should have
- `message` (*string*) – text that will be shown to the user in case of failure

**возвращает** true (правда) или false (ложь).

**тип возвращаемого значения** boolean (логический)

```
test:iscdata(slab_info.quota_size, ffi.typeof('uint64_t'), 'memcached.slab.info().
↪quota_size returns a cdata')
```

taptest.strict
Set `taptest.strict=true` if *taptest:is()* and *taptest:isnt()* and *taptest:is_ deeply()* must be compared strictly with `nil`. Set `taptest.strict=false` if `nil` and `box.NULL` both have the same effect.

The default is false. For example, if and only if `taptest.strict=true` has happened, then `taptest:is_deeply({a = box.NULL}, {})` will return `false`.

Since v. 2.8.3, `taptest.strict` is inherited in all subtests:

```
t = require('tap').test('123')
t.strict = true

t:is_deeply({a = box.NULL}, {}) -- false

t:test('subtest', function(t)
    t:is_deeply({a = box.NULL}, {}) -- also false
end)
```

### Пример

Для выполнения данного примера поместите скрипт в файл под названием ./tap.lua, затем сделайте tap.lua выполняемым файлом с помощью команды `chmod a+x ./tap.lua`, а затем выполните его, используя Tarantool в качестве обработчика скриптов после выполнения команды ./tap.lua.

```
#!/usr/bin/tarantool
local tap = require('tap')
test = tap.test("my test name")
test:plan(2)
test:ok(2 * 2 == 4, "2 * 2 is 4")
test:test("some subtests for test2", function(test)
    test:plan(2)
    test:is(2 + 2, 4, "2 + 2 is 4")
    test:isnt(2 + 3, 4, "2 + 3 is not 4")
end)
test:check()
```

Результатом вышеприведенного скрипта будет примерно следующее:

```
TAP version 13
1..2
ok - 2 * 2 is 4
    # Some subtests for test2
    1..2
    ok - 2 + 2 is 4,
    ok - 2 + 3 is not 4
    # Some subtests for test2: end
ok - some subtests for test2
```

### 9.5.35 Модуль tarantool

Выполнив команду `require('tarantool')`, можно получить ответы на вопросы о том, как был собран Tarantool-сервер, например, какие флаги были использованы, или какая версия компилятора использовалась.

Кроме того, можно проверить время работы и версию сервера, а также идентификатор процесса. Эту информацию также можно получить с помощью *box.info()*, но рекомендуется использовать модуль tarantool.

**Пример:**

```
tarantool> tarantool = require('tarantool')
---
...
tarantool> tarantool
---
- version: 2.10.4-0-g816000e
  build:
    target: Darwin-x86_64-Release
    options: cmake . -DCMAKE_INSTALL_PREFIX=/usr/local/Cellar/tarantool/2.10.4 -DENABLE_
↪BACKTRACE=ON
    linking: dynamic
    mod_format: dylib
    flags: ' -fexceptions -funwind-tables -fno-common -fopenmp -msse2 -Wformat -Wformat-
↪security
      -Werror=format-security -fstack-protector-strong -fPIC -fmacro-prefix-map=/tmp/
↪tarantool-20221113-6655-1clb1lj/tarantool-2.10.4=.
      -std=c11 -Wall -Wextra -Wno-strict-aliasing -Wno-char-subscripts -Wno-format-
↪truncation
      -Wno-gnu-alignof-expression -Wno-cast-function-type'
    compiler: Clang-14.0.0.14000029
  pid: 'function: 0x0102df34f8'
  package: Tarantool
  uptime: 'function: 0x0102df34c0'
...
tarantool> tarantool.pid()
---
- 30155
...
tarantool> tarantool.uptime()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
- 108.64641499519
...
```

### 9.5.36 Модуль uuid

#### Общие сведения

UUID означает универсальный уникальный идентификатор. Если приложение требует, чтобы значение было уникальным в рамках одной машины или одной БД, лучше использовать простой счетчик, а не UUID. Это связано с тем, что получать UUID-значения времязатратно и требует системного вызова. Используйте UUID для кластеров машин и крупных распределенных приложений. Tarantool генерирует UUID-значения по варианту 1 версии 4 правил, определенных в RFC 4122.

#### Справочник по API

Ниже приведен перечень всех функций и элементов модуля `uuid`.

| Имя | Назначение |
|---|---|
| *uuid.NULL* | Объект с нулевым UUID |
| *uuid() uuid.bin() uuid.str()* | Получение UUID |
| *uuid.new()* | Создание UUID |
| *uuid.fromstr() uuid.frombin()* *uuid_object:bin()* *uuid_object:str()* | Получение конвертированного UUID |
| *uuid.is_uuid()* | Проверка, является ли указанное значение UUID |
| *uuid_object:isnil()* | Проверка, состоит ли UUID из одних нулей |

`uuid.NULL`
A nil UUID object. Contains the all-zero UUID value – `00000000-0000-0000-0000-000000000000`.

`uuid.new()`
Since version *2.4.1*. Create a UUID sequence. You can use it in an index over a *UUID field*. For example, to create such index for a space named `test`, say:

```
tarantool> box.space.test:create_index("pk", {parts={{field = 1, type = 'uuid'}}})
```

Теперь можно вставить UUID'ы в спейс:

```
tarantool> box.space.test:insert{uuid.new()}
---
- [e631fdcc-0e8a-4d2f-83fd-b0ce6762b13f]
...

tarantool> box.space.test:insert{uuid.fromstr('64d22e4d-ac92-4a23-899a-e59f34af5479
↪')}
---
- [64d22e4d-ac92-4a23-899a-e59f34af5479]
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> box.space.test:select{}
---
- - [64d22e4d-ac92-4a23-899a-e59f34af5479]
- [e631fdcc-0e8a-4d2f-83fd-b0ce6762b13f]
...
```

**возвращает** UUID

**тип возвращаемого значения** cdata.

`uuid.__call()`

**возвращает** UUID

**тип возвращаемого значения** cdata.

`uuid.bin(`[*byte-order*]`)`

**Параметры**

- `byte-order` (*string*) – Byte order of the resulting UUID:
  - `'l'` – little-endian
  - `'b'` – big-endian
  - `'h'`, `'host'` – endianness depends on host (default)
  - `'n'`, `'network'` – endianness depends on network

**возвращает** UUID

**тип возвращаемого значения** 16-байтная строка

`uuid.str()`

**возвращает** UUID

**тип возвращаемого значения** 36-байтная двоичная строка

`uuid.fromstr(`*uuid-str*`)`

**Параметры**

- `uuid-str` (*string*) – UUID в 36-байтной шестнадцатеричной строке

**возвращает** конвертированный UUID

**тип возвращаемого значения** cdata.

`uuid.frombin(`*uuid-bin*[, *byte-order*]`)`

**Параметры**

- `uuid-bin` (*string*) – UUID в 16-байтной двоичной строке
- `byte-order` (*string*) – Byte order of the given string:
  - `'l'` – little-endian,

- `'b'` – big-endian,
- `'h'`, `'host'` – endianness depends on host (default),
- `'n'`, `'network'` – endianness depends on network.

**возвращает** конвертированный UUID

**тип возвращаемого значения** cdata.

`uuid.is_uuid(`*value*`)`

Since version *2.6.1*.

**Параметры**

- `value` – проверяемое значение

**возвращает** `true`, если указанное значение имеет тип UUID, в противном случае `false`

**тип возвращаемого значения** bool (логический)

`object uuid_object`

`uuid_object:bin(`[*byte-order*]`)`

**Параметры**

- `byte-order` (*string*) – Byte order of the resulting UUID:
  - `'l'` – little-endian,
  - `'b'` – big-endian,
  - `'h'`, `'host'` – endianness depends on host (default),
  - `'n'`, `'network'` – endianness depends on network.

**возвращает** UUID, сконвертированный из введенного значения формата cdata.

**тип возвращаемого значения** 16-байтная двоичная строка

`uuid_object:str()`

**возвращает** UUID, сконвертированный из введенного значения формата cdata.

**тип возвращаемого значения** 36-байтная шестнадцатеричная строка

`uuid_object:isnil()`

Значение UUID из одних нулей может быть выражено как *uuid.NULL* или `uuid.fromstr('00000000-0000-0000-0000-000000000000')`. Сравнение со значением из одних нулей также может быть выражено как `uuid_with_type_cdata == uuid.NULL`.

**возвращает** true (правда), если значение состоит из одних нулей, в противном случае false (ложь).

**тип возвращаемого значения** bool (логический)

#### Пример

```
tarantool> uuid = require('uuid')
---
...
tarantool> uuid(), uuid.bin(), uuid.str()
---
- 16ffedc8-cbae-4f93-a05e-349f3ab70baa
- !!binary FvG+Vy1MfUC6kIyeM81DYw==
- 67c999d2-5dce-4e58-be16-ac1bcb93160f
...
tarantool> uu = uuid()
---
...
tarantool> #uu:bin(), #uu:str(), type(uu), uu:isnil()
---
- 16
- 36
- cdata
- false
...
```

### 9.5.37 Модуль utf8

#### Общие сведения

`utf8` – это модуль Tarantool для обработки строк в формате UTF-8. Он содержит некоторые функции, которые совместимы с функциями Lua 5.3, но возможности Tarantool намного больше. Например, поскольку Tarantool включает в себя полную копию библиотеки Международных компонентов для Юникода («International Components For Unicode»), доступны также функции сравнения, которые понимают упорядочение символов в кириллице (заглавная буква Ж = строчная буква ж) и японском языке (A в хирагане = A в катакане).

| Имя | Назначение |
|---|---|
| *casecmp* and *cmp* | Сравнения |
| *lower* and *upper* | Замена регистра |
| *isalpha*, *isdigit*, *islower* and *isupper* | Определение типа символа |
| *sub* | Подстроки |
| *len* | Длина в символах |
| *next* | Посимвольная итерация |

`utf8.casecmp(`*UTF8-string*, *utf8-string*`)`

**Параметры**

- `string` (`UTF8-string`) – строка в формате UTF-8

**возвращает** -1 означает «меньше», 0 означает «равно», +1 означает «больше»

**тип возвращаемого значения** число

Сравнение двух строк с Таблицей сортировки символов Юникода по умолчанию (DUCET) для Алгоритма сортировки по Юникоду (Unicode Collation Algorithm). В результате „å“ меньше, чем

„B“, хотя значение кодовой точки å (229) больше значения кодовой точки B (66), поскольку алгоритм основывается на значениях Таблица сортировки символов, а не на значениях кодовых точек.

Сравнение осуществляется на основании основного веса. Таким образом, не учитываются элементы, которые влияют на вторичный или последующий вес (такие как «регистр» в латинице или кириллице, или «отличия каны» в японском языке). Если спросить: «Это похоже на сортировку без учета регистра и ударения от компании Майкрософт?» - ответом будет: «Скорее да», хотя Алгоритм сортировки по Юникоду гораздо сложнее, чем это описание.

**Пример:**

```
tarantool> utf8.casecmp('é','e'),utf8.casecmp('E','e')
---
- 0
- 0
...
```

`utf8.char(`*code-point*[, *code-point* ...]`)`

**Параметры**

- `number` (`code-point`) – значение кодовой точки в Юникоде, повторяется

**возвращает** строка в UTF-8

**тип возвращаемого значения** строка

Число кодовой точки – это значение, которое соответствует символу в Базе данных символов Юникода This is not the same as the byte values of the encoded character, because the UTF-8 encoding scheme is more complex than a simple copy of the code-point number.

Другой способ создать строку с символами Юникода – с помощью механизма экранирования символов \u{шестнадцатеричные-числа}, например, в результате и „\u{41}\u{42}“, и `utf8.char(65, 66)` получим строку „AB“.

**Пример:**

```
tarantool> utf8.char(229)
---
- å
...
```

`utf8.cmp(`*UTF8-string*, *utf8-string*`)`

**Параметры**

- `string` (`UTF8-string`) – строка в формате UTF-8

**возвращает** -1 означает «меньше», 0 означает «равно», +1 означает «больше»

**тип возвращаемого значения** число

Сравнение двух строк с Таблицей сортировки символов Юникода по умолчанию (DUCET) для Алгоритма сортировки по Юникоду (Unicode Collation Algorithm). В результате „å“ меньше, чем „B“, хотя значение кодовой точки å (229) больше значения кодовой точки B (66), поскольку алгоритм основывается на значениях Таблица сортировки символов, а не на значениях кода.

Сравнение осуществляется на основании не менее трех значений веса. Таким образом, не учитываются элементы, которые влияют на вторичный или последующий вес (такие как «регистр» в

латинице или кириллице, или «отличия каны» в японском языке), а верхний регистр следует за нижним.

**Пример:**

```
tarantool> utf8.cmp('é','e'),utf8.cmp('E','e')
---
- 1
- 1
...
```

utf8.isalpha(*UTF8-character*)

**Параметры**

- `string-or-number` (`UTF8-character`) – отдельный символ UTF8, выраженный в виде однобайтной строки или значения кодовой точки

**возвращает** true (правда) или false (ложь)

**тип возвращаемого значения** boolean (логический)

Возврат true (правда), если введенный символ является буквенным, в остальных случаях – false (ложь). В целом, символ считается буквенным, если он используется в рамках слова, а не как число или знак пунктуации. Такой символ необязательно должен быть буквой алфавита.

**Пример:**

```
tarantool> utf8.isalpha('Ж'),utf8.isalpha('å'),utf8.isalpha('9')
---
- true
- true
- false
...
```

utf8.isdigit(*UTF8-character*)

**Параметры**

- `string-or-number` (`UTF8-character`) – отдельный символ UTF8, выраженный в виде однобайтной строки или значения кодовой точки

**возвращает** true (правда) или false (ложь)

**тип возвращаемого значения** boolean (логический)

Возврат true (правда), если введенный символ является цифрой, в остальных случаях – false (ложь).

**Пример:**

```
tarantool> utf8.isdigit('Ж'),utf8.isdigit('å'),utf8.isdigit('9')
---
- false
- false
- true
...
```

utf8.islower(*UTF8-character*)

**Параметры**

- string-or-number (UTF8-character) – отдельный символ UTF8, выраженный в виде однобайтной строки или значения кодовой точки

**возвращает** true (правда) или false (ложь)

**тип возвращаемого значения** boolean (логический)

Возврат true (правда), если введенный символ относится к нижнему регистру, в остальных случаях – false (ложь).

**Пример:**

```
tarantool> utf8.islower('Ж'),utf8.islower('å'),utf8.islower('9')
---
- false
- true
- false
...
```

utf8.isupper(*UTF8-character*)

**Параметры**

- string-or-number (UTF8-character) – отдельный символ UTF8, выраженный в виде однобайтной строки или значения кодовой точки

**возвращает** true (правда) или false (ложь)

**тип возвращаемого значения** boolean (логический)

Возврат true (правда), если введенный символ относится к верхнему регистру, в остальных случаях – false (ложь).

**Пример:**

```
tarantool> utf8.isupper('Ж'),utf8.isupper('å'),utf8.isupper('9')
---
- true
- false
- false
...
```

utf8.len(*UTF8-string*[, *start-byte*[, *end-byte*]])

**Параметры**

- string (UTF8-string) – строка в формате UTF-8
- integer (end-byte) – позиция байта первого символа
- integer – позиция байта для остановки

**возвращает** количество символов в строке или же от начала до конца

**тип возвращаемого значения** число

Позиции байта в начале и в конце могут быть отрицательными, что указывает на отсчет с конца строки, а не с начала.

Если строка содержит последовательность байтов, которая неприменима для UTF-8, каждый байт в неправильной последовательности будет считаться за один символ.

UTF-8 представляет собой схему кодирования изменяемого размера. Как правило, одна буква латиницы занимает один байт, буква кириллицы занимает два байта, а символ из китайского или японского языка занимает три байта, максимальный размер – четыре байта.

**Пример:**

```
tarantool> utf8.len('G'),utf8.len('ж')
---
- 1
- 1
...

tarantool> string.len('G'),string.len('ж')
---
- 1
- 2
...
```

`utf8.lower(`*UTF8-string*`)`

**Параметры**

- `string` (`UTF8-string`) – строка в формате UTF-8

**возвращает** та же строка в нижнем регистре

**тип возвращаемого значения** строка

**Пример:**

```
tarantool> utf8.lower('ÅЖABCDEFG')
---
- åжabcdefg
...
```

`utf8.next(`*UTF8-string*`[,` *start-byte* `])`

**Параметры**

- `string` (`UTF8-string`) – строка в формате UTF-8
- `integer` (`start-byte`) – позиция байта внутри строки, с которой начать выполнение, по умолчанию = 1

**возвращает** позиция байта следующего символа и значение кодовой точки следующего символа

**тип возвращаемого значения** таблица

Функция `next` часто используется в цикле для получения символа за раз из строки в формате UTF-8.

**Пример:**

В строке „åa“ первый символ – „å“, он начинается в позиции 1, занимает два байта, поэтому символ после него будет на позиции 3, значение кодовой точки в Юникоде (десятичное) – 229.

```
tarantool> -- показать позицию следующего символа + кодовую точку первого символа
tarantool> utf8.next('åa', 1)
---
- 3
- 229
...
tarantool> -- (цикл) показать кодовую точку каждого символа
tarantool> for position,codepoint in utf8.next,'åa' do print(codepoint) end
229
97
...
```

utf8.sub(*UTF8-string*, *start-character*[, *end-character*])

**Параметры**

- string (UTF8-string) – строка в формате UTF-8
- number (end-character) – позиция первого символа
- number – позиция последнего символа

**возвращает** строка в формате UTF-8, «подстрока» введенного значения

**тип возвращаемого значения** строка

Позиции символа в начале и в конце могут быть отрицательными, что указывает на отсчет с конца строки, а не с начала.

Значение end-character по умолчанию – длина введенной строки. Таким образом, выполнение `utf8.sub(1, 'abc')` вернет „abc“, т.е. введенную строку.

**Пример:**

```
tarantool> utf8.sub('åжabcdefg', 5, 8)
---
- abcd
...
```

utf8.upper(*UTF8-string*)

**Параметры**

- string (UTF8-string) – строка в формате UTF-8

**возвращает** та же строка в верхнем регистре

**тип возвращаемого значения** строка

---

**Примечание:** В редких случаях результат в верхнем регистре может быть длиннее введенной строки в нижнем регистре, например, `utf8.upper('ß')` вернет „SS“.

---

**Пример:**

```
tarantool> utf8.upper('åжabcdefg')
---
- ÅЖABCDEFG
...
```

### 9.5.38 Модуль uri

#### Общие сведения

The URI module provides functions that convert URI strings into their components, or turn components into URI strings, for example:

```
local uri = require('uri')

parsed_uri = uri.parse('https://www.tarantool.io/doc/latest/reference/reference_lua/http/
↪#api-reference')
--[[
---
- host: www.tarantool.io
  fragment: api-reference
  scheme: https
  path: /doc/latest/reference/reference_lua/http/
...
--]]

formatted_uri = uri.format({ scheme = 'https',
                             host = 'www.tarantool.io',
                             path = '/doc/latest/reference/reference_lua/http/',
                             fragment = 'api-reference' })
--[[
---
- https://www.tarantool.io/doc/latest/reference/reference_lua/http/#api-reference
...
--]]
```

You can also use this module to encode and decode arbitrary strings using the specified encoding options.

### API Reference

Below is a list of `uri` functions, properties, and related objects.

| **Functions** | |
|---|---|
| *uri.parse()* | Получение таблицы URI-компонентов |
| *uri.format()* | Construct a URI from the specified components |
| *uri.escape()* | Encode a string using the specified encoding options |
| *uri.unescape()* | Decode a string using the specified encoding options |
| **Properties** | |
| *uri.RFC3986* | Encoding options that use unreserved symbols defined in RFC 3986 |
| *uri.PATH* | Options used to encode the `path` URI component |
| *uri.PATH_PART* | Options used to encode specific `path` parts |
| *uri.QUERY* | Options used to encode the `query` URI component |
| *uri.QUERY_PART* | Options used to encode specific `query` parts |
| *uri.FRAGMENT* | Options used to encode the `fragment` URI component |
| *uri.FORM_URLENCODED* | Options used to encode `application/x-www-form-urlencoded` form parameters |
| **Related objects** | |
| *uri_components* | URI components |
| *uri_encoding_opts* | URI encoding options |

### Functions

`uri.parse(`*uri-string*`)`
Parse a URI string into components.

**See also:** *uri.format()*

**Параметры**

- `uri-string` (*string*) – a URI string

**возвращает** a URI components table (see *uri_components*)

**тип возвращаемого значения** *table*

**Пример:**

```
local uri = require('uri')

parsed_uri = uri.parse('https://www.tarantool.io/doc/latest/reference/reference_lua/
↪http/#api-reference')
--[[
---
- host: www.tarantool.io
  fragment: api-reference
  scheme: https
  path: /doc/latest/reference/reference_lua/http/
...
--]]
```

`uri.format(`*uri_components*`[`, *include_password*`])`
Construct a URI from the specified components.

**See also:** *uri.parse()*

**Параметры**

- `uri_components` (*table*) – a series of `name=value` pairs, one for each component (see *uri_components*)
- `include_password` (`boolean`) – specify whether the password component is rendered in clear text; otherwise, it is omitted

**возвращает** URI string

**тип возвращаемого значения** строка

**Пример:**

```
local uri = require('uri')

formatted_uri = uri.format({ scheme = 'https',
                             host = 'www.tarantool.io',
                             path = '/doc/latest/reference/reference_lua/http/',
                             fragment = 'api-reference' })
--[[
---
- https://www.tarantool.io/doc/latest/reference/reference_lua/http/#api-reference
...
--]]
```

`uri.escape(`*string*[, *uri_encoding_opts*]`)`

**Since:** *2.11.0*

Encode a string using the specified encoding options.

By default, `uri.escape()` uses encoding options defined by the *uri.RFC3986* table. If required, you can customize encoding options using the `uri_encoding_opts` optional parameter, for example:

- Pass the predefined set of options targeted for encoding a specific URI part (for example, *uri.PATH* or *uri.QUERY*).
- Pass custom encoding options using the *uri_encoding_opts* object.

**Параметры**

- `string` – a string to encode
- `uri_encoding_opts` (*table*) – encoding options (optional, see *uri_encoding_opts*)

**возвращает** an encoded string

**тип возвращаемого значения** строка

**Example 1:**

This example shows how to encode a string using the default encoding options.

```
local uri = require('uri')

escaped_string = uri.escape('C++')
--[[
---
- C%2B%2B
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
--]]
```

**Example 2:**

This example shows how to encode a string using the *uri.FORM_URLENCODED* encoding options.

```
local uri = require('uri')

escaped_string_url_enc = uri.escape('John Smith', uri.FORM_URLENCODED)
--[[
---
- John+Smith
...
--]]
```

**Example 3:**

This example shows how to encode a string using custom encoding options.

```
local uri = require('uri')

local escape_opts = {
    plus = true,
    unreserved = uri.unreserved("a-z")
}
escaped_string_custom = uri.escape('Hello World', escape_opts)
--[[
---
- '%48ello+%57orld'
...
--]]
```

uri.unescape(*string*[, *uri_encoding_opts*])

**Since:** *2.11.0*

Decode a string using the specified encoding options.

By default, `uri.escape()` uses encoding options defined by the *uri.RFC3986* table. If required, you can customize encoding options using the `uri_encoding_opts` optional parameter, for example:

- Pass the predefined set of options targeted for encoding a specific URI part (for example, *uri.PATH* or *uri.QUERY*).
- Pass custom encoding options using the *uri_encoding_opts* object.

**Параметры**

- `string` – a string to decode
- `uri_encoding_opts` (*table*) – encoding options (optional, see *uri_encoding_opts*)

**возвращает** a decoded string

**тип возвращаемого значения** строка

**Example 1:**

This example shows how to decode a string using the default encoding options.

```
local uri = require('uri')

unescaped_string = uri.unescape('C%2B%2B')
--[[
---
- C++
...
--]]
```

**Example 2:**

This example shows how to decode a string using the *uri.FORM_URLENCODED* encoding options.

```
local uri = require('uri')

unescaped_string_url_enc = uri.unescape('John+Smith', uri.FORM_URLENCODED)
--[[
---
- John Smith
...
--]]
```

**Example 3:**

This example shows how to decode a string using custom encoding options.

```
local uri = require('uri')

local escape_opts = {
    plus = true,
    unreserved = uri.unreserved("a-z")
}
unescaped_string_custom = uri.unescape('%48ello+%57orld', escape_opts)
--[[
---
- Hello World
...
--]]
```

### Properties

uri.RFC3986
: Encoding options that use unreserved symbols defined in RFC 3986. These are default options used to encode and decode using the *uri.escape()* and *uri.unescape()* functions, respectively.

**See also:** *uri_encoding_opts*

**тип возвращаемого значения** table

uri.PATH
: Options used to encode the `path` URI component.

**See also:** *uri_encoding_opts*

**тип возвращаемого значения** table

`uri.PATH_PART`
Options used to encode specific `path` parts.

**See also:** *uri_encoding_opts*

**тип возвращаемого значения** table

`uri.QUERY`
Options used to encode the `query` URI component.

**See also:** *uri_encoding_opts*

**тип возвращаемого значения** table

`uri.QUERY_PART`
Options used to encode specific `query` parts.

**See also:** *uri_encoding_opts*

**тип возвращаемого значения** table

`uri.FRAGMENT`
Options used to encode the `fragment` URI component.

**See also:** *uri_encoding_opts*

**тип возвращаемого значения** table

`uri.FORM_URLENCODED`
Options used to encode `application/x-www-form-urlencoded` form parameters.

**See also:** *uri_encoding_opts*

**тип возвращаемого значения** table

### Related objects

#### uri_components

`object uri_components`
URI components. The `uri_components` object is used in the following functions:

- The *uri.parse()* function returns the `uri_components` object.
- The *uri.format()* function accepts the `uri_components` object as an argument.

`uri_components.scheme`
A URI scheme.

**Examples:** `https`, `http`

`uri_components.login`
A user name, which is a part of the `userinfo` subcomponent.

`uri_components.password`
A password, which is a part of the `userinfo` subcomponent.

`uri_components.host`
A host subcomponent.

**Example:** `www.tarantool.io`

`uri_components.service`
A service subcomponent. This property might return different values depending on the used URI scheme, for example:

- If the `https` or `http` scheme is used, `service` returns the port value.
- If the Unix domain socket is used, `service` returns the socket path.

**Examples:** `3301`, `/tmp/unix.sock`

`uri_components.path`
A path component.

**Example:** `/doc/latest/reference/reference_lua/http/`

`uri_components.query`
A query component.

**Example:** `key1=value1&key2=value2`

`uri_components.fragment`
A fragment component.

**Example:** `api-reference`

`uri_components.ipv4`
An IPv4 address.

**Example:** `127.0.0.1`

`uri_components.ipv6`
An IPv6 address.

**Example:** `2a00:1148:b0ba:2016:12bf:48ff:fe78:fd10`

`uri_components.unix`
A Unix domain socket.

**Example:** `/tmp/unix.sock`

### uri_encoding_opts

`object uri_encoding_opts`
**Since:** *2.11.0*

URI encoding options. These options can be passed to the *uri.escape()* and *uri.unescape()* functions.

**Пример:**

The example below shows how to encode a string using custom encoding options.

```
local uri = require('uri')

local escape_opts = {
    plus = true,
    unreserved = uri.unreserved("a-z")
}
escaped_string_custom = uri.escape('Hello World', escape_opts)
--[[
---
- '%48ello+%57orld'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
--]]
```

**Примечание:** The `uri` module also provides several sets of predefined options targeted for encoding a specific URI part (for example, *uri.PATH* or *uri.QUERY*).

`uri_encoding_opts.plus`
Enable encoding of `+` as the space character. By default, this property is set to `false`.

**тип возвращаемого значения** boolean

`uri_encoding_opts.unreserved`
Specify a Lua pattern defining unreserved symbols that are not encoded.

**тип возвращаемого значения** table

**Example:** `'a-zA-Z0-9%-._~'`

### 9.5.39 Модуль xlog

Модуль xlog включает в себя одну функцию: `pairs()`. Ее можно использовать для чтения *файлов снимка* или *файлов журнала упреждающей записи (WAL)* в Tarantool. Описание формата файла дается в разделе *Персистентность данных и формат WAL-файла*.

`xlog.pairs(`[*file-name*]`)`
Открытие файла и итерация по одной записи файла за раз.

**возвращает** итератор, который можно использовать в цикле for / end.

**тип возвращаемого значения** итератор

Возможные ошибки: Файл не содержит снимок в правильном формате или информацию журнала упреждающей записи.

**Пример:**

В данном примере производится чтение первого WAL-файла, который был создан в директории *wal_dir* в рамках наших *упражнений в «Руководстве для начинающих»*.

Каждый результат из `pairs()` выводится в формате MsgPack, поэтому его структуру можно указать с помощью *__serialize*.

```
xlog = require('xlog')
t = {}
for k, v in xlog.pairs('00000000000000000000.xlog') do
  table.insert(t, setmetatable(v, { __serialize = "map"}))
end
return t
```

Первые строки результата будут выглядеть следующим образом:

```
(...)
---
- - {'BODY':   {'space_id': 272, 'index_base': 1, 'key': ['max_id'],
                'tuple': [['+', 2, 1]]},
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    'HEADER': {'type': 'UPDATE', 'timestamp': 1477846870.8541,
               'lsn': 1, 'server_id': 1}}
- {'BODY':   {'space_id': 280,
              'tuple': [512, 1, 'tester', 'memtx', 0, {}, []]},
   'HEADER': {'type': 'INSERT', 'timestamp': 1477846870.8597,
              'lsn': 2, 'server_id': 1}}
```

### 9.5.40 Модуль yaml

#### Общие сведения

The `yaml` module takes strings in YAML format and decodes them, or takes a series of non-YAML values and encodes them.

#### Указатель

Ниже приведен перечень всех функций и элементов модуля `yaml`.

| Имя | Назначение |
|---|---|
| *yaml.encode()* | Конвертация Lua-объекта в YAML-строку |
| *yaml.decode()* | Конвертация YAML-строки в Lua-объект |
| *__serialize parameter* | Output structure specification |
| *yaml.cfg()* | Изменение конфигурации |
| *yaml.NULL* | Аналог «nil» в языке Lua |

`yaml.encode(`*lua_value*`)`
Конвертация Lua-объекта в YAML-строку.

**Параметры**

- `lua_value` – скалярное значение или значение из Lua-таблицы.

**возвращает** оригинальное значение, преобразованное в YAML-строку.

**тип возвращаемого значения** строка

`yaml.decode(`*string*`)`
Конвертация YAML-строки в Lua-объект.

**Параметры**

- `string` – строка в формате YAML.

**возвращает** оригинальное содержание в формате Lua-таблицы.

**тип возвращаемого значения** таблица

**__serialize parameter:**

The YAML output structure can be specified with `__serialize`:

- `'seq'`, `'sequence'`, `'array'`: table encoded as an array
- `'map'`, `'mapping'`: table encoded as a map
- `function`: the meta-method called to unpack serializable representation of table, cdata, or userdata objects

'seq' or 'map' also enable the flow (compact) mode for the YAML serializer (flow="[1,2,3]" vs block="
- 1\n - 2\n - 3\n").

Serializing 'A' and 'B' with different __serialize values brings different results:

```
tarantool> yaml.encode(setmetatable({'A', 'B'}, { __serialize="seq"}))
---
- '["A","B"]'
...
tarantool> yaml.encode(setmetatable({'A', 'B'}, { __serialize="map"}))
---
- '{"1":"A","2":"B"}'
...
tarantool> yaml.encode({setmetatable({f1 = 'A', f2 = 'B'}, { __serialize="map"})})
---
- '[{"f2":"B","f1":"A"}]'
...
tarantool> yaml.encode({setmetatable({f1 = 'A', f2 = 'B'}, { __serialize="seq"})})
---
- '[[]]'
...
```

yaml.cfg(*table*)

Set values affecting the behavior of encode and decode functions.

The values are all either integers or boolean true/false.

| Характеристика | Значение по умолчанию | Назначение |
|---|---|---|
| cfg.encode_invalid_numbers | true | A flag saying whether to enable encoding of NaN and Inf numbers |
| cfg.encode_number_precision | 14 | Precision of floating point numbers |
| cfg.encode_load_metatables | true | A flag saying whether the serializer will follow *__serialize* metatable field |
| cfg.encode_use_tostring | false | A flag saying whether to use tostring() for unknown types |
| cfg.encode_invalid_as_nil | false | A flag saying whether to use NULL for non-recognized types |
| cfg.encode_sparse_convert | true | A flag saying whether to handle excessively sparse arrays as maps. See detailed description *below* |
| cfg.encode_sparse_ratio | 2 | 1/encode_sparse_ratio is the permissible percentage of missing values in a sparse array |
| cfg.encode_sparse_safe | 10 | A limit ensuring that small Lua arrays are always encoded as sparse arrays (instead of generating an error or encoding as map) |
| cfg.decode_invalid_numbers | true | A flag saying whether to enable decoding of NaN and Inf numbers |
| cfg.decode_save_metatables | true | A flag saying whether to set metatables for all arrays and maps |

**Sparse arrays features:**

During encoding, The YAML encoder tries to classify table into one of four kinds:

- Map: at least one table index is not unsigned integer.

- Regular array: all array indexes are available.
- Sparse array: at least one array index is missing.
- Excessively sparse array: the number of values missing exceeds the configured ratio.

An array is excessively sparse when **all** the following conditions are met:

- `encode_sparse_ratio` > 0
- `max(table)` > `encode_sparse_safe`
- `max(table)` > `count(table)` * `encode_sparse_ratio`

The YAML encoder will never consider an array to be excessively sparse when `encode_sparse_ratio` = 0. The `encode_sparse_safe` limit ensures that small Lua arrays are always encoded as sparse arrays. By default, attempting to encode an excessively sparse array will generate an error. If `encode_sparse_convert` is set to `true`, excessively sparse arrays will be handled as maps.

**yaml.cfg() example 1:**

The following code will encode 0/0 as NaN («not a number») and 1/0 as Inf («infinity»), rather than returning nil or an error message:

```
yaml = require('yaml')
yaml.cfg{encode_invalid_numbers = true}
x = 0/0
y = 1/0
yaml.encode({1, x, y, 2})
```

Результат запроса `yaml.encode()` будет следующим:

```
tarantool> yaml.encode({1, x, y, 2})
---
- '[1,nan,inf,2]
...
```

**yaml.cfg example 2:**

To avoid generating errors on attempts to encode unknown data types as userdata/cdata, you can use this code:

```
tarantool> httpc = require('http.client').new()
---
...

tarantool> yaml.encode(httpc.curl)
---
- error: unsupported Lua type 'userdata'
...

tarantool> yaml.encode(httpc.curl, {encode_use_tostring=true})
---
- '"userdata: 0x010a4ef2a0"'
...
```

**Примечание:** To achieve the same effect for only one call to `yaml.encode()` (i.e. without changing the configuration permanently), you can use `yaml.encode({1, x, y, 2}, {encode_invalid_numbers =`

true}).

Similar configuration settings exist for *JSON* and *MsgPack*.

yaml.NULL
: Значение, сопоставимое с нулевым значением «nil» в языке Lua, которое можно использовать в качестве объекта-заполнителя в кортеже.

### Пример

```
tarantool> yaml = require('yaml')
---
...
tarantool> y = yaml.encode({'a', 1, 'b', 2})
---
...
tarantool> z = yaml.decode(y)
---
...
tarantool> z[1], z[2], z[3], z[4]
---
- a
- 1
- b
- 2
...
tarantool> if yaml.NULL == nil then print('hi') end
hi
---
...
```

Набор YAML-стилей можно указать с помощью `__serialize`:

- `__serialize="sequence"` для массива последовательности блоков,
- `__serialize="seq"` для массива последовательности потоков,
- `__serialize="mapping"` для ассоциативного массива последовательности блоков,
- `__serialize="map"` для ассоциативного массива последовательности потоков.

Serializing `'A'` and `'B'` with different `__serialize` values causes different results:

```
tarantool> yaml = require('yaml')
---
...

tarantool> print(yaml.encode(setmetatable({'A', 'B'}, { __serialize="sequence"})))
---
- A
- B
...

---
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tarantool> print(yaml.encode(setmetatable({'A', 'B'}, { __serialize="seq"})))
--- ['A', 'B']
...

---
...

tarantool> print(yaml.encode({setmetatable({f1 = 'A', f2 = 'B'}, { __serialize="map"})}))
---
- {'f2': 'B', 'f1': 'A'}
...

---
...
```

### 9.5.41 Прочие компоненты пакета

All the Tarantool modules are, at some level, inside a package which, appropriately, is named `package`. There are also miscellaneous functions and variables which are outside all modules.

| Имя | Назначение |
|---|---|
| *tonumber64()* | Конвертация строки или Lua-числа в 64-битное целое число |
| *dostring()* | Анализ и выполнение произвольного Lua-кода |
| *package.path* | Get file paths used to search for Lua modules |
| *package.cpath* | Get file paths used to search for C modules |
| *package.loaded* | Show Lua or C modules loaded by Tarantool |
| *package.searchroot* | Get the root path for a directory search |
| *package.setsearchroot* | Set the root path for a directory search |

`tonumber64`(*value*)

Конвертация строки или Lua-числа в 64-битное целое число. Входное значение может быть выражено десятичным, двоичным (например, 0b1010) или шестнадцатеричным (например, -0xffff) числом. Результат может использоваться в арифметике, причем скорее в 64-битной целочисленной арифметике, а не в арифметике в системе с плавающей запятой. (Операции с неконвертированными Lua-числами выполняются в арифметике в системе с плавающей запятой.) Функция `tonumber64()` в Tarantool является глобальной.

**Пример:**

```
tarantool> type(123456789012345), type(tonumber64(123456789012345))
---
- number
- number
...
tarantool> i = tonumber64('1000000000')
---
...
tarantool> type(i), i / 2, i - 2, i * 2, i + 2, i % 2, i ^ 2
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- number
- 500000000
- 999999998
- 2000000000
- 1000000002
- 0
- 1000000000000000000
...
```

**Warning:** There is an underlying LuaJIT library that operates with C rules. Therefore you should expect odd results if you compare unsigned and signed (for example 0ULL > -1LL is false), or if you use numbers outside the 64-bit integer range (for example 9223372036854775808LL is negative). Also you should be aware that `type(`*`number-literal-ending-in-ULL`*`)` is cdata, not a Lua arithmetic type, which prevents direct use with some functions in Lua libraries such as math. See the LuaJIT reference and look for the phrase «64 bit integer arithmetic». and the phrase «64 bit integer comparison». Or see the comments on Issue#4089.

`dostring(`*lua-chunk-string*[, *lua-chunk-string-argument ...*]`)`

Анализ и выполнение произвольного Lua-кода. Данная функция используется преимущественно для определения и выполнения Lua-кода без необходимости внесения изменений в глобальное Lua-окружение.

**Параметры**

- `lua-chunk-string` (*string*) – Lua-код
- `lua-chunk-string-argument` (`lua-value`) – ноль или другие скалярные значения, которые заменяются или к которым прибавляются значения.

**возвращает** то, что возвращает Lua-код.

Возможные ошибки: Ошибка компиляции появляется как Lua-ошибка.

**Пример:**

```
tarantool> dostring('abc')
---
error: '[string "abc"]:1: ''='' expected near ''<eof>'''
...
tarantool> dostring('return 1')
---
- 1
...
tarantool> dostring('return ...', 'hello', 'world')
---
- hello
- world
...
tarantool> dostring([[
         >   local f = function(key)
         >     local t = box.space.tester:select{key}
         >     if t ~= nil then
         >       return t[1]
         >     else
         >       return nil
```

(continues on next page)

(продолжение с предыдущей страницы)

```
         >     end
         >   end
         >   return f(...)]], 1)
---
- null
...
```

`package.path`
: Get file paths used to search for Lua *modules*. For example, these paths are used to find modules loaded using the `require()` directive.

  See also: *package.searchroot()*

`package.cpath`
: Get file paths used to search for C *modules*. For example, these paths are used to find modules loaded using the `require()` directive.

  See also: *package.searchroot()*

`package.loaded`
: Show Lua or C modules loaded by Tarantool, so that their functions and members are available. `loaded` shows both pre-loaded modules and modules added using the `require()` directive.

  See also: *package.searchroot()*

`package.searchroot()`
: Return the current search root, which defines the path to the root directory from which dependencies are loaded. By default, the search root is the current directory.

---

**Примечание:** The current directory is obtained using *debug.sourcedir()*.

---

**Example**

Suppose the application has the following structure:

```
/home/testuser/myapp
├── .rocks/share/tarantool/
│   └── foo.lua
├── init.lua
└── modules
    └── bar.lua
```

In this case, modules are placed in the same directory as the application initialization file. If you *run the application* using the `tarantool` command from the `myapp` directory, ...

```
/home/testuser/myapp$ tarantool init.lua
```

... the search root is `/home/testuser/myapp` and Tarantool finds all modules in this directory automatically. This means that to load the `foo` and `modules.bar` modules in `init.lua`, you only need to add the corresponding `require` directives:

```
-- init.lua --
require('foo')
require('modules.bar')
```

Starting with *2.11.0*, you can also run the application using the `tarantool` command from the directory other than `myapp`:

```
/home/testuser$ tarantool myapp/init.lua
```

In this case, the path to the initialization file (`/home/testuser/myapp`) is added to search paths for modules.

To load modules placed outside of the path to the application directory, use *package.setsearchroot()*.

`package.setsearchroot(`[*search-root*]`)`

Set the search root, which defines the path to the root directory from which dependencies are loaded. By default, the search root is the current directory (see *package.searchroot()*).

**Параметры**

- `search-root` (*string*) – a relative or absolute path to the search root. If `search-root` is a relative path, it is expanded to an absolute path. You can omit this argument or set it to *box.NULL* to reset the search root to the current directory.

**Example**

Suppose external modules are stored outside the application directory, for example:

```
/home/testuser/
├── myapp
│   └── init.lua
└── mymodules
    ├── .rocks/share/tarantool/
    │   └── foo.lua
    └── modules
        └── bar.lua
```

In this case, you can specify the `/home/testuser/mymodules` path as the search root for modules in the following way:

```
-- init.lua --
package.setsearchroot('/home/testuser/mymodules')
```

Then, you can load the `foo` and `bar` modules using the `require()` directive:

```
-- init.lua --
require('foo')
require('modules.bar')
```

### 9.5.42 Коды ошибок базы данных

The table below lists some popular errors that can be raised by Tarantool in case of various issues. You can find a complete list of errors in the errcode.h file.

---

**Примечание:** The *box.error* module provides the ability to get the information about the last error raised by Tarantool or raise custom errors manually.

---

| Code | box.error value | Description |
|---|---|---|
| ER_NONMASTER | box.error.NONMASTER | (Репликация) Экземпляр сервера не может вносить изменения в данные, если он не является мастером. |
| ER_ILLEGAL_PARAMS | box.error.ILLEGAL_PARAMS | Недопустимые параметры. Некорректное протокольное сообщение. |
| ER_MEMORY_ISSUE | box.error.MEMORY_ISSUE | Нехватка оперативной памяти: достижение предела памяти *memtx_memory*. |
| ER_WAL_IO | box.error.WAL_IO | Failed to write to disk. May mean: failed to record a change in the write-ahead log. |
| ER_READONLY | box.error.READONLY | Can't modify data on a read-only instance. |
| ER_KEY_PART_COUNT | box.error.KEY_PART_COUNT | Key part count is not the same as index part count. |
| ER_NO_SUCH_SPACE | box.error.NO_SUCH_SPACE | Указанный спейс отсутствует. |
| ER_NO_SUCH_INDEX | box.error.NO_SUCH_INDEX | Указанного индекса нет в указанном спейсе. |
| ER_PROC_LUA | box.error.PROC_LUA | Возникла ошибке в Lua-процедуре. |
| ER_FIBER_STACK | box.error.FIBER_STACK | При создании нового файбера был достигнут предел рекурсии. Обычно это указывает на то, что хранимая процедура слишком часто рекурсивно вызывает себя. |
| ER_UPDATE_FIELD | box.error.UPDATE_FIELD | Возникла ошибка во время обновления поля. |
| ER_TUPLE_FOUND | box.error.TUPLE_FOUND | В уникальном индексе есть повторяющийся ключ. |

### Обработка ошибок

Ниже представлены несколько процедур для более надежного вызова Lua-функций в случае ошибок, в частности, ошибок базы данных.

1. Invoke a function using `pcall`.

   Take advantage of Lua's mechanisms for Error handling and exceptions, particularly `pcall`. That is, instead of invoking with . . .

   ```
   box.space.{space-name}:{function-name}()
   ```

   . . . call the function as follows:

   ```
   if pcall(box.space.{space-name}.{function-name}, box.space.{space-name}) ...
   ```

   For some Tarantool box functions, `pcall` also returns error details, including a file-name and line-number within Tarantool's source code. This can be seen by unpacking, for example:

   ```
   status, error = pcall(function() box.schema.space.create('') end)
   error:unpack()
   ```

   See the tutorial *Sum a JSON field for all tuples* to see how `pcall` can fit in an application.

2. Examine errors and raise new errors using `box.error`.

To make a new error and pass it on, the `box.error` module provides *box.error()*.

To find the last error, the `box.error` submodule provides *box.error.last()*. There is also a way to find the text of the last operating-system error for certain functions – *errno.strerror([code])*.

3. Запись в журнал.

   Записывайте сообщения в журнал с помощью *модуля log*.

   Filter automatically generated messages using the *log* configuration parameter.

Generally, for Tarantool built-in functions which are designed to return objects: the result is an object, or nil, or a Lua error. For example consider the *fio_read.lua* program in a cookbook:

```
#!/usr/bin/env tarantool

local fio = require('fio')
local errno = require('errno')
local f = fio.open('/tmp/xxxx.txt', {'O_RDONLY' })
if not f then
    error("Failed to open file: "..errno.strerror())
end
local data = f:read(4096)
f:close()
print(data)
```

After a function call that might fail, like `fio.open()` above, it is common to see syntax like `if not f then ...` or `if f == nil then ...`, which check for common failures. But if there had been a syntax error, for example fio.opex instead of fio.open, then there would have been a Lua error and f would not have been changed. If checking for such an obvious error had been a concern, the programmer would probably have used `pcall()`.

Все функции в модулях Tarantool должны работать таким образом, если в руководстве явно не говорится об обратном.

### 9.5.43 Средства отладки

#### Общие сведения

Пользователи Tarantool могут воспользоваться преимуществами встроенных средств отладки, которые составляют часть:

- Lua (библиотека отладки, см. подробное описание ниже) и
- LuaJit (функции отладки debug.*).

Библиотека `debug` предоставляет интерфейс для отладки Lua-программ. Все функции этой библиотеки содержатся в таблице `debug`. В функциях для работы с потоками есть дополнительный первый параметр, в котором указывается необходимый поток. По умолчанию, это всегда текущий поток.

---

**Примечание:** Библиотеку следует использовать только для отладки и профилирования, а не в качестве программного средства, поскольку данные функции выполняются слишком долго. Кроме того, некоторые из этих функций могут привести к нарушению работы безопасного в других отношениях кода.

---

### Указатель

Ниже приведен перечень всех функций библиотеки `debug`.

| Имя | Назначение |
|---|---|
| *debug.debug()* | Вход в интерактивный режим |
| *debug.getfenv()* | Получение среды объекта |
| *debug.gethook()* | Получение текущих настроек ловушки потока |
| *debug.getinfo()* | Получение информации о функции |
| *debug.getlocal()* | Получение имени и значения локальной переменной |
| *debug.getmetatable()* | Получение метатаблицы объекта |
| *debug.getregistry()* | Получение таблицы реестра |
| *debug.getupvalue()* | Получение имени и значения сопоставляющего значения |
| *debug.setfenv()* | Определение среды объекта |
| *debug.sethook()* | Определение данной функции в качестве ловушки |
| *debug.setlocal()* | Присваивание значения локальной переменной |
| *debug.setmetatable()* | Определение метатаблицы объекта |
| *debug.setupvalue()* | Присваивание значения сопоставляющему значению |
| *debug.sourcedir()* | Get the source directory name |
| *debug.sourcefile()* | Get the source file name |
| *debug.traceback()* | Получение обратной трассировки стека вызовов |

`debug.debug()`

Вход в интерактивный режим и выполнение каждой строки, которую печатает пользователь. Пользователь может, в частности, проверять глобальные и локальные переменные, изменять их значения и вычислять выражения.

Введите `cont` для выхода из данной функции, чтобы вызывающий клиент мог продолжить выполнение.

---

**Примечание:** Команды для `debug.debug()` не вложены лексически в какую-либо функцию, поэтому у них нет прямого доступа к локальным переменным.

---

`debug.getfenv(`*object*`)`

**Параметры**

- `object` – объект, для которого будет получена среда

**возвращает** среда объекта `object`

`debug.gethook(`[*thread*]`)`

**возвращает** текущие настройки ловушки потока `thread` в виде трех значений:

- текущая функция-ловушка
- текущая маска ловушки
- текущий счетчик ловушки, как определяет функция `debug.sethook()`

`debug.getinfo(`[*thread*]`,` *function*[`,` *what*]`)`

**Параметры**

- `function` – функция, по которой будет получена информация
- `what` (*string*) – какую информацию о функции `function` вернуть

**возвращает** таблица с информацией о функции `function`

Можно передать функцию `function` напрямую или же передать число, которое указывает на функцию, выполняемую на уровне `function` стека вызовов данного потока `thread`: уровень 0 – это текущая функция (сама функция `getinfo()`), уровень 1 – это функция, которая вызвала `getinfo()`, и т.д. Если для функции `function` указано число больше числа активных функций, `getinfo()` вернет `nil`.

По умолчанию, `what` – это вся доступная информация, кроме таблицы допустимых строк. Если задать опцию `f`, добавится поле под названием `func` с самой функцией. Если задать опцию `L`, добавится поле под названием `activelines` с таблицей доступных строк.

`debug.getlocal(`[*thread*]`, `*level*`, `*local*`)`

**Параметры**

- `level` (`number`) – уровень стека
- `local` (`number`) – индекс локальной переменной

**возвращает** имя и значение локальной переменной с индексом `local` функции на уровне `level` стека или `nil`, если нет локальной переменной с указанным индексом; появится ошибка, если уровень `level` вне диапазона

---

**Примечание:** Можно вызвать `debug.getinfo()` для проверки доступности уровня.

---

`debug.getmetatable(`*object*`)`

**Параметры**

- `object` – объект, для которого будет получена метатаблица

**возвращает** метатаблица объекта `object` или `nil`, если метатаблица отсутствует

`debug.getregistry()`

**возвращает** таблица реестра

`debug.getupvalue(`*func*`, `*up*`)`

**Параметры**

- `func` (`function`) – функция, для которой будет получено сопоставляющее значение
- `up` (`number`) – индекс сопоставляющего значения функции

**возвращает** имя и значение сопоставляющего значения с индексом `up` функции `func` или `nil`, если нет сопоставляющего значения в пределах заданного индекса

`debug.setfenv(`*object*`, `*table*`)`
Определение среды объекта `object` для таблицы `table`.

**Параметры**

- `object` – объект, среда которого будет изменена

- `table` (*table*) – таблица для определения среды объекта

**возвращает** объект `object`

`debug.sethook`([*thread*], *hook*, *mask*[, *count*])

Определение данной функции в качестве ловушки. При вызове без аргументов ловушка отключается.

**Параметры**

- `hook` (`function`) – функция, которая будет определена в качестве ловушки
- `mask` (*string*) – описание того, когда будет вызвана ловушка `hook`; может принимать следующие значения: * `c` – ловушка``hook`` вызывается каждый раз, когда Lua вызывает функцию * `r` – ловушка `hook` вызывается каждый раз, когда Lua возвращается из функции * `l` – ловушка `hook` вызывается каждый раз, когда Lua переходит на новую строку кода
- `count` (`number`) – описание того, когда будет вызвана ловушка `hook`; если отличается от нуля, ловушка `hook` вызывается после каждой инструкции `count`.

`debug.setlocal`([*thread*], *level*, *local*, *value*)

Присвоение значения `value` локальной переменной с индексом `local` функции на уровне `level` стека

**Параметры**

- `level` (`number`) – уровень стека
- `local` (`number`) – индекс локальной переменной
- `value` – значение, присваиваемое локальной переменной

**возвращает** имя локальной переменной или `nil`, если локальная переменная с заданным индексом отсутствует; возникает ошибка, если уровень `level` вне диапазона

---

**Примечание:** Можно вызвать `debug.getinfo()` для проверки доступности уровня.

---

`debug.setmetatable`(*object*, *table*)

Определение метатаблицы объекта `object` для таблицы `table`.

**Параметры**

- `object` – объект, метатаблица которого будет изменена
- `table` (*table*) – таблица для определения метатаблицы объекта

`debug.setupvalue`(*func*, *up*, *value*)

Присвоение значения `value` сопоставляющему значению с индексом `up` функции `func`.

**Параметры**

- `func` (`function`) – функция, для которой будет определено сопоставляющее значение
- `up` (`number`) – индекс сопоставляющего значения функции
- `value` – значение, присваиваемое сопоставляющему значению функции

**возвращает** имя сопоставляющего значения или `nil`, если сопоставляющее значение с данным индексом отсутствует

`debug.sourcedir(`[*level*]`)`

**Параметры**

- `level` (`number`) – the level of the call stack which should contain the path (default is 2)

**возвращает** a string with the relative path to the source file directory

Instead of `debug.sourcedir()` one can say `debug.__dir__` which means the same thing.

Determining the real path to a directory is only possible if the function was defined in a Lua file (this restriction may not apply for loadstring() since Lua will store the entire string in debug info).

If `debug.sourcedir()` is part of a `return` argument, then it should be inside parentheses: `return (debug.sourcedir())`.

`debug.sourcefile(`[*level*]`)`

**Параметры**

- `level` (`number`) – the level of the call stack which should contain the path (default is 2)

**возвращает** a string with the relative path to the source file

Instead of `debug.sourcefile()` one can say `debug.__file__` which means the same thing.

Determining the real path to a file is only possible if the function was defined in a Lua file (this restriction may not apply to `loadstring()` since Lua will store the entire string in debug info).

If `debug.sourcefile()` is part of a `return` argument, then it should be inside parentheses: `return (debug.sourcefile())`.

`debug.traceback(`[*thread*]`,` [*message*]`,` [*level*]`)`

**Параметры**

- `message` (*string*) – необязательное сообщение, добавленное к началу обратной трассировки
- `level` (`number`) – указывает на каком уровне начинать обратную трассировку (по умолчанию, 1)

**возвращает** строка с обратной трассировкой стека вызовов

**Debug example:**

Make a file in the /tmp directory named example.lua, containing:

```
function w()
  print(debug.sourcedir())
  print(debug.sourcefile())
  print(debug.traceback())
  print(debug.getinfo(1)['currentline'])
end
w()
```

Execute `tarantool /tmp/example.lua`. Expect to see this:

```
/tmp
/tmp/example.lua
stack traceback:
    /tmp/example.lua:4: in function 'w'
    /tmp/example.lua:7: in main chunk
5
```

### 9.5.44 JSON-пути

#### Общие сведения

Since version *2.3*, Tarantool supports JSON path updates. You can update or upsert formatted *tuple* / *space* / *index* fields by name (not only by field number). Updates of nested structures are also supported.

**Пример:**

```
tarantool> box.cfg{};
         > format = {};
         > format[1] = {'field1', 'unsigned'};
         > format[2] = {'field2', 'map'};
         > format[3] = {'field3', 'array'};
         > format[4] = {'field4', 'string', is_nullable = true}
---
...
tarantool> s = box.schema.create_space('test', {format = format});
         > _ = s:create_index('pk')
---
...
tarantool> t = {
         >     1,
         >     {
         >         key1 = 'value',
         >         key2 = 10
         >     },
         >     {
         >         2,
         >         3,
         >         {key3 = 20}
         >     }
         > }
---
...
tarantool> t = s:replace(t)
---
...
tarantool> t:update({{'=', 'field2.key1', 'new_value'}})
---
- [1, {'key1': 'new_value', 'key2': 10}, [2, 3, {'key3': 20}]]
...
tarantool> t:update({{'+', 'field3[2]', 1}})
---
- [1, {'key1': 'value', 'key2': 10}, [2, 4, {'key3': 20}]]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...
tarantool> s:update({1}, {{'!', 'field4', 'inserted value'}})
---
- [1, {'key1': 'value', 'key2': 10}, [2, 3, {'key3': 20}], 'inserted value']
...
tarantool> s:update({1}, {{'#', '[2].key2', 1}, {'=', '[3][3].key4', 'value4'}})
---
- [1, {'key1': 'value'}, [2, 3, {'key3': 20, 'key4': 'value4'}], 'inserted value']
...
tarantool> s:upsert({1, {k = 'v'}, {}}, {{'#', '[2].key1', 1}})
---
...
tarantool> s:select{}
---
- - [1, {}, [2, 3, {'key3': 20, 'key4': 'value4'}], 'inserted value']
...
```

Обратите внимание, что имена полей, которые выглядят как JSON-пути, обрабатываются аналогично *доступу к полям кортежа через JSON*: сначала весь путь интерпретируется как имя поля; если такого имени не существует, то оно обрабатывается как путь.

Например, для имя поля `field.name.like.json`, это обновление

*object-name*`:update(...,` *`'field.name.like.json'`*`, ...)`

обновит именно это поле целиком, а не ключи `field` -> `name` -> `like` -> `json`. Если это имя нужно вам как часть большего пути, то его нужно обернутьв кавычки `""` или квадратные скобки `[]`:

*object-name*`:update(...,` *`'["field.name.like.json"].next.fields'`*`, ...)`

**Есть несколько правил для обновления через JSON:**

- Операция `'!'` не может быть использована для создания всех промежуточных узлов пути.Например, `{'!', 'field1[1].field3', ...}` не может создать поля `'field1'` и `'[1]'`, они должны существовать.
- Операция `'#'`, при применении к ассоциативным массивам, не может удалить сразу несколько ключей. Поэтому для ассоциативных массивов ее аргумент всегда должен быть 1.

  `{'#', 'field1.field2', 1}` – разрешено;

  `{'#', 'field1.field2', 10}` – не разрешено.

  Это ограничение возникает из-за проблемы, что ключи на ассоциативном массиве все равно не упорядочены, а `'#'` с более чем 1 ключом приведет к неопределенному поведению.
- Операция `'!'` на ассоциативных массивах не может создать ключ, если он уже существует.
- Если ассоциативный массив содержит нестроковые ключи (булеаны, цифры, ассоциативные массивы, массивы - все что угодно), то эти ключи не могут быть обновлены с помощью JSON-путей. Но в таком ассоциативном массиве все равно разрешено обновлять строковые ключи.

**Почему обновления с помощью JSON-путей хороши, и их следует предпочитать, когда нужно обновить только часть кортежа:**

- Они расходуют меньше места в WAL, потому что для обновления хранятся только ключи, операции и аргументы. Обновление одного труднодоступного поля дешевле, чем обновление всего кортежа.

- Они быстрее. Во-первых, это потому, что они реализованы на C, и у них нет проблем с Lua GC и динамическим набором текста. Во-вторых, некоторые случаи использования JSON-путей хорошо оптимизированы. Например, обновление одним JSON-путем стоит O(1) памяти, независимо от того, насколько глубоко этот путь проходит (не считая аргументов обновления).
- Они доступны с удаленных клиентов, а также с любого другого DML. До того, как обновления с помощью JSON-путей стали доступны в Tarantool, чтобы обновить одну глубокую часть кортежа, нужно было скачать этот кортеж, обновить его в памяти и отправить обратно — 2 сетевых хопа. Если применяются JSON-пути, обновление может быть описано в путях и занять 1 хоп.

## 9.6 Справочник по сторонним библиотекам

В данном справочнике описаны сторонние Lua-модули для Tarantool.

### 9.6.1 Модуль membership

Этот модуль представляет собой библиотеку `membership` для Tarantool на основе протокола gossip.

Эта библиотека создает сеть из нескольких экземпляров Tarantool. Сеть сама контролирует себя, помогает участникам обнаружить всех остальных в группе и получать уведомления об изменениях своего статуса с низкой задержкой. Модуль основан на концепциях из Consul или, точнее, алгоритма SWIM.

Модуль `membership` работает по протоколу UDP и может производить операции даже до инициализации box.cfg.

#### Структура членов данных

Члены-данные представлены в виде таблиц со следующими полями:

- `uri` (строка) – это унифицированный идентификатор ресурса.
- `status` (строка) – это строка, которая принимает одно из следующих значений.
  - `alive`: член группы, который отвечает на сообщения проверки связи, работоспособен в статусе `alive`.
  - `suspect`: если какой-либо член группы не может получить ответ от какого-либо другого участника, первый член группы просит трех других активных членов группы в статусе `alive` отправить сообщение проверки связи соответствующему участнику. Если ответа нет, последний получает статус сомнительного, то есть `suspect`.
  - `dead`: член группы в статусе `suspect` получает статус вышедшего из строя `dead` по истечении времени ожидания.
  - `left`: член группы получает статус выбывшего `left` после выполнения функции *leave()*.

    **Примечание:** Протокол gossip гарантирует, что каждый член группы узнает о любом изменении статуса в двух циклах связи.

- `incarnation` (число) – это значение, которое увеличивается каждый раз, когда экземпляр получает статус `suspect`, `dead` или обновляет полезную нагрузку.
- `payload` (таблица) – это вспомогательные данные, которыми могут воспользоваться различные модули.

- `timestamp` (число) – это значение `fiber.time64()`, которое:
  - соответствует последнему обновлению параметра `status` или `incarnation`;
  - всегда локально;
  - не зависит от настроек часов других членов группы.

Ниже приведен пример таблицы:

```
tarantool> membership.myself()
---
uri: localhost:33001
status: alive
incarnation: 1
payload:
    uuid: 2d00c500-2570-4019-bfcc-ab25e5096b73
timestamp: 1522427330993752
...
```

### Справочник по API

Ниже приведен список простых функций, функций шифрования, подписки и параметры модуля `membership`.

| Имя | Назначение |
|---|---|
| Простые функции | |
| *init(advertise_host, port)* | Инициализация модуля `membership`. |
| *myself()* | Получение структуры данных текущего экземпляра. |
| *get_member(uri)* | Получение структуры данных для указанного URI. |
| *members()* | Получение таблицы со всеми членами группы, известными текущему экземпляру. |
| *pairs()* | Сокращение для `pairs(membership.members())`. |
| *add_member(uri)* | Добавление члена в группу. |
| *probe_uri(uri)* | Проверка принадлежности члена к группе. |
| *broadcast()* | Обнаружение участников в локальной сети путем отправки широковещательного сообщения UDP. |
| *set_payload(key, value)* | Обновление `myself().payload` и распространение информации. |
| *leave()* | Корректное исключение из группы. |
| *is_encrypted()* | Проверка, включено ли шифрование. |
| Функции шифрования | |
| *set_encryption_key(key)* | Установка ключа для низкоуровневого шифрования сообщений. |
| *get_encryption_key()* | Получение используемого ключа шифрования. |
| Функции подписки | |
| *subscribe()* | Подписка на обновления членов таблицы. |
| *unsubscribe()* | Удаление подписки. |
| Параметры | |
| *PROTOCOL_PERIOD_SECONDS* | Время отправки сообщений проверки связи напрямую. |
| *ACK_TIMEOUT_SECONDS* | Время ожидания сообщения подтверждения. |
| *ANTI_ENTROPY_PERIOD_SECONDS* | Период синхронизации во избежание энтропии. |
| *SUSPECT_TIMEOUT_SECONDS* | Время ожидания, чтобы перевести члена группы из статуса `suspect` в `dead`. |
| *NUM_FAILURE_DETECTION_SUBGROUPS* | Число членов группы, которые отправляют сообщения проверки связи члену группы в статусе `suspect`. |

Простые функции:

membership.init(*advertise_host*, *port*)
: Инициализация модуля membership. Привязывает UDP-сокет к 0.0.0.0:<port>, задает значение параметра advertise_uri = <advertise_host>:<port> (передаваемый хост, порт) и значение параметра incarnation = 1.

  Функцию init() можно вызвать несколько раз, старый сокет будет закрыт, откроется новый сокет.

  Если значение параметра advertise_uri изменится во время очередного выполнения init(), старый URI считается недоступным со статусом DEAD. Чтобы корректно исключить члена из группы, используйте функцию *leave()*.

  **Параметры**

  - advertise_host (*string*) – имя хоста или IP-адрес, передаваемый другим членам группы
  - port (number) – привязываемый UDP-порт

  **возвращает** true (правда)

  **тип возвращаемого значения** boolean (логический)

  **вызывает** подключенный сокет, если нет ошибки

membership.myself()

  **возвращает** *структура данных члена группы* для текущего экземпляра.

  **тип возвращаемого значения** таблица

membership.get_member(*uri*)

  **Параметры**

  - uri (*string*) – advertise_uri для указанного члена группы

  **возвращает** *структура данных* экземпляра с указанным URI.

  **тип возвращаемого значения** таблица

membership.members()
: Получение всех членов группы, известных текущему экземпляру.

  Редактирование этой таблицы ни на что не вляет.

  **возвращает** таблица с URI в качестве ключей и *структурой данных члена группы* в качестве значений.

  **тип возвращаемого значения** таблица

membership.pairs()
: Сокращение для pairs(membership.members()).

  **возвращает** Lua-итератор

  Можно использовать следующим образом:

```
for uri, member in membership.pairs()
  -- что-то сделать
end
```

membership.add_member(*uri*)
Добавление в группу члена с указанным URI и передача информации об этом событии другим членам группы. Достаточно добавить члена группы в один экземпляр, так как все остальные экземпляры в группе со временем получат информацию об этом. Не имеет значения, кто кого добавляет.

**Параметры**

- uri (*string*) – параметр advertise_uri добавляемого члена группы

**возвращает** true (правда) или нулевое значение nil в случае ошибки

**тип возвращаемого значения** boolean (логический)

**вызывает** ошибка анализа, если URI нельзя проанализировать

membership.probe_uri(*uri*)
Отправка сообщения члену группы, чтобы убедиться, что он включен в группу. Если экземпляр активен со статусом alive, но не включен в группу, происходит его добавление. Если он уже включен в группу, ничего не происходит.

**Параметры**

- uri (*string*) – параметр advertise_uri члена группы, которому отправляются сообщения проверки связи

**возвращает** true (правда), если ответ возвращается в течение 0.2 секунды, в остальных случаях no response (нет ответа)

**тип возвращаемого значения** boolean (логический)

**вызывает** ping was not sent (сообщение проверки связи не отправлено), если имя хоста не разрешено

membership.broadcast()
Обнаружение членов группы в локальной сети путем отправки широковещательного сообщения UDP во все сети, обнаруженные с помощью вызова getifaddrs() на языке C.

**возвращает** true (правда), если сообщение отправлено, false (ложь), если getaddrinfo() не выполнена.

**тип возвращаемого значения** boolean (логический)

membership.set_payload(*key*, *value*)
Обновление myself().payload и распространение соответствующей информации вместе со статусом члена группы.

Увеличивает значение параметра incarnation.

**Параметры**

- key (*string*) – ключ, задаваемый в таблице payload
- value – дополнительные данные

**возвращает** true (правда)

**тип возвращаемого значения** boolean (логический)

membership.leave()
Корректное исключение из группы membership. Узел получает статус выбывшего left, другие члены группы не будут пытаться снова подключить его.

**возвращает** true (правда)

**тип возвращаемого значения** boolean (логический)

membership.is_encrypted()

**возвращает** `true` (правда), если шифрование включено, `false` в противном случае.

**тип возвращаемого значения** boolean (логический)

Функции шифрования:

membership.set_encryption_key(*key*)
Установка ключа, который используется для низкоуровневого шифрования сообщений. Ключ автоматически обрезается или дополняется до 32 байтов. Если значения ключа `key` нулевое `nil`, шифрование будет отключено.

Модуль Tarantool `crypto.cipher.aes256.cbc` занимается шифрованием.

Чтобы обеспечить правильную связь, все члены группы должны быть настроены на использование одного и того же ключа шифрования. В противном случае члены группы получат статус либо `dead`, либо `non-decryptable` (невозможно расшифровать).

**Параметры**

- `key` (*string*) – ключ шифрования

**возвращает** `nil`.

membership.get_encryption_key()
Получение используемого ключа шифрования.

**возвращает** ключ шифрования или нулевое значение `nil`, если шифрование отключено.

**тип возвращаемого значения** строка

Функции подписки:

membership.subscribe()
Подписка на обновления членов таблицы.

**возвращает** объект `fiber.cond`, который передается при каждом изменении таблицы.

**тип возвращаемого значения** объект

membership.unsubscribe(*cond*)
Удаление подписки на `cond`, получаемый с помощью функции *subscribe()*.

Достоверность `cond` не проверяется.

**Параметры**

- `cond` – объект `fiber.cond`, получаемый с помощью функции *subscribe()*

**возвращает** `nil`.

Ниже приведен перечень параметров `membership`. Их можно задать следующим образом:

```
options = require('membership.options')
options.<параметр> = <значение>
```

options.PROTOCOL_PERIOD_SECONDS
Период отправки сообщение проверки связи напрямую. Обозначается как `T'` в протоколе SWIM.

options.ACK_TIMEOUT_SECONDS
Время ожидания сообщения подтверждения после отправки сообщения проверки связи. Если ответ запаздывает, вызывается алгоритм косвенной проверки связи.

`options.ANTI_ENTROPY_PERIOD_SECONDS`
: Период выполнения алгоритма синхронизации во избежание энтропии из протокола SWIM.

`options.SUSPECT_TIMEOUT_SECONDS`
: Время ожидания, чтобы перевести члена группы из статуса `suspect` в `dead`.

`options.NUM_FAILURE_DETECTION_SUBGROUPS`
: Число членов группы, которые пытаются отправить сообщения проверки связи члену группы в статусе `suspect`. Обозначается как `k` в протоколе SWIM.

### 9.6.2 Luatest

More about Luatest API see *below*.

#### Overview

Tool for testing tarantool applications.

Highlights:

- executable to run tests in directory or specific files,
- before/after suite hooks,
- before/after test group hooks,
- *output capturing*,
- *helpers* for testing tarantool applications,
- *luacov integration*.

#### Requirements

- Tarantool (it requires tarantool-specific `fio` module and `ffi` from LuaJIT).

#### Installation

```
tt rocks install luatest
.rocks/bin/luatest --help # list available options
```

#### Usage

Define tests.

```
-- test/feature_test.lua
local t = require('luatest')
local g = t.group('feature')
-- Default name is inferred from caller filename when possible.
-- For `test/a/b/c_d_test.lua` it will be `a.b.c_d`.
-- So `local g = t.group()` works the same way.

-- Tests. All properties with name staring with `test` are treated as test cases.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
g.test_example_1 = function() ... end
g.test_example_n = function() ... end

-- Define suite hooks
t.before_suite(function() ... end)
t.before_suite(function() ... end)

-- Hooks to run once for tests group
g.before_all(function() ... end)
g.after_all(function() ... end)

-- Hooks to run for each test in group
g.before_each(function() ... end)
g.after_each(function() ... end)

-- Hooks to run for a specified test in group
g.before_test('test_example_1', function() ... end)
g.after_test('test_example_2', function() ... end)
-- before_test runs after before_each
-- after_test runs before after_each

-- test/other_test.lua
local t = require('luatest')
local g = t.group('other')
-- ...
g.test_example_2 = function() ... end
g.test_example_m = function() ... end

-- Define parametrized groups
local pg = t.group('pgroup', {{engine = 'memtx'}, {engine = 'vinyl'}})
pg.test_example_3 = function(cg)
    -- Use cg.params here
    box.schema.space.create('test', {
        engine = cg.params.engine,
    })
end

-- Hooks can be specified for one parameter
pg.before_all({engine = 'memtx'}, function() ... end)
pg.before_each({engine = 'memtx'}, function() ... end)
pg.before_test('test_example_3', {engine = 'vinyl'}, function() ... end)
```

Run tests from a path.

```
luatest                              # run all tests from the ./test directory
luatest test/integration             # run all tests from the specified directory
luatest test/feature_test.lua        # run all tests from the specified file
```

Run tests from a group.

```
luatest feature                      # run all tests from the specified group
luatest other.test_example_2         # run one test from the specified group
luatest feature other.test_example_2 # run tests by group and test name
```

Note that luatest recognizes an input parameter as a path only if it contains /, otherwise, it will be considered as a group name.

```
luatest feature                         # considered as a group name
luatest ./feature                       # considered as a path
luatest feature/                        # considered as a path
```

You can also use `-p` option in combination with the examples above for running tests matching to some name pattern.

```
luatest feature -p test_example         # run all tests from the specified group matching␣
↪to the specified pattern
```

Luatest automatically requires `test/helper.lua` file if it's present. You can configure luatest or run any bootstrap code there.

See the getting-started example in cartridge-cli repo.

### Tests order

Use the `--shuffle` option to tell luatest how to order the tests. The available ordering schemes are `group`, `all` and `none`.

`group` shuffles tests within the groups.

`all` randomizes execution order across all available tests. Be careful: `before_all/after_all` hooks run always when test group is changed, so it may run multiple time.

`none` is the default, which executes examples within the group in the order they are defined (eventually they are ordered by functions line numbers).

With `group` and `all` you can also specify a `seed` to reproduce specific order.

```
--shuffle none
--shuffle group
--shuffle all --seed 123
--shuffle all:123 # same as above
```

To change default order use:

```
-- test/helper.lua
local t = require('luatest')
t.configure({shuffle = 'group'})
```

### List of luatest functions

| Assertions | |
|---|---|
| assert (value[, message]) | Check that value is truth |
| assert_almost_equals (actual, expected, margin[, message]) | Check that two floats are |
| assert_covers (actual, expected[, message]) | Checks that actual map |
| assert_lt (left, right[, message]) | Compare numbers. |
| assert_le (left, right[, message]) | |
| assert_gt (left, right[, message]) | |

Таблица 6 – продолжение с предыдущей страницы

| | |
|---|---|
| assert_ge (left, right[, message]) | |
| assert_equals (actual, expected[, message[, deep_analysis]]) | Check that two values ar |
| assert_error (fn, ...) | Check that calling fn rais |
| assert_error_msg_contains (expected_partial, fn, ...) | |
| assert_error_msg_content_equals (expected, fn, ...) | Strips location info from |
| assert_error_msg_equals (expected, fn, ...) | Checks full error: locatio |
| assert_error_msg_matches (pattern, fn, ...) | |
| assert_error_covers (expected, fn, ...) | Checks that actual error |
| assert_eval_to_false (value[, message]) | Alias for assert_not. |
| assert_eval_to_true (value[, message]) | Alias for assert. |
| assert_items_include (actual, expected[, message]) | Checks that one table inc |
| assert_is (actual, expected[, message]) | Check that values are th |
| assert_is_not (actual, expected[, message]) | Check that values are no |
| assert_items_equals (actual, expected[, message]) | Checks that two tables c |
| assert_nan (value[, message]) | |
| assert_not (value[, message]) | Check that value is falsy. |
| assert_not_almost_equals (actual, expected, margin[, message]) | Check that two floats are |
| assert_not_covers (actual, expected[, message]) | Checks that map does no |
| assert_not_equals (actual, expected[, message]) | Check that two values ar |
| assert_not_nan (value[, message]) | |
| assert_not_str_contains (actual, expected[, is_pattern[, message]]) | Case-sensitive strings cor |
| assert_not_str_icontains (value, expected[, message]) | Case-insensitive strings c |
| assert_str_contains (value, expected[, is_pattern[, message]]) | Case-sensitive strings cor |
| assert_str_icontains (value, expected[, message]) | Case-insensitive strings c |
| assert_str_matches (value, pattern[, start=1[, final=value:len() [, message]]]) | Verify a full match for th |
| assert_type (value, expected_type[, message]) | Check value's type. |
| **Flow control** | |
| fail (message) | Stops a test due to a fail |
| fail_if (condition, message) | Stops a test due to a fail |
| xfail (message) | Mark test as xfail. |
| xfail_if (condition, message) | Mark test as xfail if cond |
| skip (message) | Skip a running test. |
| skip_if (condition, message) | Skip a running test if cor |
| success () | Stops a test with a succe |
| success_if (condition) | Stops a test with a succe |
| **Suite and groups** | |
| after_suite (fn) | Add after suite hook. |
| before_suite (fn) | Add before suite hook. |
| group (name) | Create group of tests. |

### XFail

The `xfail` mark makes test results to be interpreted vice versa: it's threated as passed when an assertion fails, and it fails if no errors are raised. It allows one to mark a test as temporarily broken due to a bug in some other component which can't be fixed immediately. It's also a good practice to keep xfail tests in sync with an issue tracker.

```
local g = t.group()
g.test_fail = function()
    t.xfail('Must fail no matter what')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    t.assert_equals(3, 4)
end
```

XFail only applies to the errors raised by the luatest assertions. Regular Lua errors still cause the test failure.

### Capturing output

By default runner captures all stdout/stderr output and shows it only for failed tests. Capturing can be disabled with `-c` flag.

### Tests repeating

Runners can repeat tests with flags `-r` / `--repeat` (to repeat all the tests) or `-R` / `--repeat-group` (to repeat all the tests within the group).

### Parametrization

Test group can be parametrized.

```
local g = t.group('pgroup', {{a = 1, b = 4}, {a = 2, b = 3}})

g.test_params = function(cg)
    ...
    log.info('a = %s', cg.params.a)
    log.info('b = %s', cg.params.b)
    ...
end
```

Group can be parametrized with a matrix of parameters using `luatest.helpers`:

```
local g = t.group('pgroup', t.helpers.matrix({a = {1, 2}, b = {3, 4}}))
-- Will run:
-- * a = 1, b = 3
-- * a = 1, b = 4
-- * a = 2, b = 3
-- * a = 2, b = 4
```

Each test will be performed for every params combination. Hooks will work as usual unless there are specified params. The order of execution in the hook group is determined by the order of declaration.

```
-- called before every test
g.before_each(function(cg) ... end)

-- called before tests when a == 1
g.before_each({a = 1}, function(cg) ... end)

-- called only before the test when a == 1 and b == 3
g.before_each({a = 1, b = 3}, function(cg) ... end)

-- called before test named 'test_something' when a == 1
```

(continues on next page)

(продолжение с предыдущей страницы)

```
g.before_test('test_something', {a = 1}, function(cg) ... end)

--etc
```

Test from a parameterized group can be called from the command line in such a way:

```
luatest pgroup.a:1.b:4.test_params
luatest pgroup.a:2.b:3.test_params
```

Note that values for `a` and `b` have to match to defined group params. The command below will give you an error because such params are not defined for the group.

```
luatest pgroup.a:2.b:2.test_params  # will raise an error
```

### Test helpers

There are helpers to run tarantool applications and perform basic interaction with it. If application follows configuration conventions it is possible to use options to configure server instance and helpers at the same time. For example `http_port` is used to perform http request in tests and passed in `TARANTOOL_HTTP_PORT` to server process.

```
local server = luatest.Server:new({
    command = '/path/to/executable.lua',
    -- arguments for process
    args = {'--no-bugs', '--fast'},
    -- additional envars to pass to process
    env = {SOME_FIELD = 'value'},
    -- passed as TARANTOOL_WORKDIR
    workdir = '/path/to/test/workdir',
    -- passed as TARANTOOL_HTTP_PORT, used in http_request
    http_port = 8080,
    -- passed as TARANTOOL_LISTEN, used in connect_net_box
    net_box_port = 3030,
    -- passed to net_box.connect in connect_net_box
    net_box_credentials = {user = 'username', password = 'secret'},
})
server:start()
-- Wait until server is ready to accept connections.
-- This may vary from app to app: for one server:connect_net_box() is enough,
-- for another more complex checks are required.
luatest.helpers.retrying({}, function() server:http_request('get', '/ping') end)

-- http requests
server:http_request('get', '/path')
server:http_request('post', '/path', {body = 'text'})
server:http_request('post', '/path', {json = {field = value}, http = {
    -- http client options
    headers = {Authorization = 'Basic ' .. credentials},
    timeout = 1,
}})
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- This method throws error when response status is outside of then range 200..299.
-- To change this behaviour, path `raise = false`:
t.assert_equals(server:http_request('get', '/not_found', {raise = false}).status, 404)
t.assert_error(function() server:http_request('get', '/not_found') end)

-- using net_box
server:connect_net_box()
server:eval('return do_something(...)', {arg1, arg2})
server:call('function_name', {arg1, arg2})
server:exec(function() return box.info() end)
server:stop()
```

`luatest.Process:start(path, args, env)` provides low-level interface to run any other application.

There are several small helpers for common actions:

```
luatest.helpers.uuid('ab', 2, 1) == 'abababab-0002-0000-0000-000000000001'

luatest.helpers.retrying({timeout = 1, delay = 0.1}, failing_function, arg1, arg2)
-- wait until server is up
luatest.helpers.retrying({}, function() server:http_request('get', '/status') end)
```

### luacov integration

- Install luacov with `tt rocks install luacov`
- Configure it with `.luacov` file
- Clean old reports `rm -f luacov.*.out*`
- Run luatest with `--coverage` option
- Generate report with `.rocks/bin/luacov .`
- Show summary with `grep -A999 '^Summary' luacov.report.out`

When running integration tests with coverage collector enabled, luatest automatically starts new tarantool instances with luacov enabled. So coverage is collected from all the instances. However this has some limitations:

- It works only for instances started with `Server` helper.
- Process command should be executable lua file or tarantool with script argument.
- Instance must be stopped with `server:stop()`, because this is the point where stats are saved.
- Don't save stats concurrently to prevent corruption.

### Development

- Check out the repo.
- Prepare makefile with `cmake ..`
- Install dependencies with `make bootstrap`.
- Run it with `make lint` before committing changes.
- Run tests with `bin/luatest`.

### Contributing

Bug reports and pull requests are welcome on at https://github.com/tarantool/luatest.

### License

MIT

### API

### Module *luatest.helpers*

Collection of test helpers.

### Functions

Methods

### matrix (parameters_values)

Return all combinations of parameters. Accepts params“ names and thier every possible value.

```
helpers.matrix({a = {1, 2}, b = {3, 4}})

{
  {a = 1, b = 3},
  {a = 2, b = 3},
  {a = 1, b = 4},
  {a = 2, b = 4},
}
```

**Parameters:**

- *parameters_values:* (**tab**)

### retrying (config, fn, ...)

Keep calling fn until it returns without error. Throws last error if config.timeout is elapsed. Default options are taken from helpers.RETRYING_TIMEOUT and helpers.RETRYING_DELAY.

```
helpers.retrying({}, fn, arg1, arg2)
helpers.retrying({timeout = 2, delay = 0.5}, fn, arg1, arg2)
```

**Parameters:**

- *config:*
  - *timeout:* (**number**)
  - *delay:* (**number**)
- *fn:* (**func**)
- *...:* args

### uuid (a, ...)

Generates uuids from its 5 parts. Strings are repeated and numbers are padded to match required part length. If number of arguments is less than 5 then first and last arguments are used for corresponding parts, missing parts are set to 0.

```
'aaaaaaaa-0000-0000-0000-000000000000' == uuid('a')
'abababab-0000-0000-0000-000000000001' == uuid('ab', 1)
'00000001-0002-0000-0000-000000000003' == uuid(1, 2, 3)
'11111111-2222-0000-0000-333333333333' == uuid('1', '2', '3')
'12121212-3434-5656-7878-909090909090' == uuid('12', '34', '56', '78', '90')
```

**Parameters:**

- *a:* first part
- *...:* parts

### Class *luatest.group*

Tests group.

To add new example add function at key starting with `test` .

Group hooks run always when test group is changed. So it may run multiple times when `--shuffle` option is used.

#### Instance methods

##### Group.mt.after_all (fn)

Add callback to run once after all tests in the group.

**Parameters:**

- *fn:*

##### Group.mt.after_each (fn)

Add callback to run after each test in the group.

**Parameters:**

- *fn:*

##### Group.mt.before_all (fn)

Add callback to run once before all tests in the group.

**Parameters:**

- *fn:*

##### Group.mt.before_each (fn)

Add callback to run before each test in the group.

**Parameters:**

- *fn:*

##### Group.mt:initialize ([name])

**Parameters:**

- *name:* (string) Default name is inferred from caller filename when possible.For `test/a/b/c_d_test.lua` it will be `a.b.c_d` . (optional)

**Returns**:

Group instance

### Class *luatest.http_response*

Class to provide helper methods for HTTP responses

#### Instance getter methods

#### HTTPResponse.getters

For backward compatibility this methods should be accessed as object's fields (eg., `response.json.id` ). They are not assigned to object's fields on initialization to be evaluated lazily and to be able to throw errors.

#### HTTPResponse.getters:json ()

Parse json from body.

#### Usage:

```
response.json.id
```

#### HTTPResponse.mt:is_successful ()

Check that status code is 2xx.

### Class *luatest.runner*

Class to run test suite.

#### Functions

Methods

#### Runner.is_test_name (s)

Check that string matches the name of a test method. Default rule is that is starts with „test“

**Parameters:**

- *s:*

### Runner.run ([args=_G.args[, options]])

Main entrypoint to run test suite.

**Parameters:**

- *args:* (**tab**) List of CLI arguments (default $(def))
- *options:*
  - *verbosity:* (**int**) (optional)
  - *fail_fast:* (**bool**) (default $(def))
  - *output_file_name:* (string) Filename for JUnit report (optional)
  - *exe_repeat:* (**int**) Times to repeat each test (optional)
  - *exe_repeat_group:* (**int**) Times to repeat each group of tests (optional)
  - *tests_pattern:* (**tab**) Patterns to filter tests (optional)
  - *tests_names:* (**tab**) List of test names or groups to run (optional)
  - *paths:* (**tab**) List of directories to load tests from. (default $(def))
  - *load_tests:* (**func**) Function to load tests. Called once for every item in `paths` . (optional)
  - *shuffle:* (string) Shuffle method (none, all, group) (default $(def))
  - *seed:* (**int**) Random seed for shuffle (optional)
  - *output:* (string) Output formatter (text, tap, junit, nil) (default $(def))

### Runner.split_test_method_name (someName)

Split `some.group.name.method` into `some.group.name` and `method` . Returns `nil, input` if input value does not have a dot.

**Parameters:**

- *someName:*

### Runner:expand_group (group)

Exrtact all test methods from group.

**Parameters:**

- *group:*

### Class *luatest.server*

Class to manage Tarantool instances.

#### Functions

Methods

#### Server.build_listen_uri (server_alias[, extra_path])

Build a listen URI based on the given server alias and extra path. The resulting URI: `<Server.vardir>/[<extra_path>/]<server_alias>.sock`. Provide a unique alias or extra path to avoid collisions with other sockets. For now, only UNIX sockets are supported.

**Parameters:**

- *server_alias:* (string) Server alias.
- *extra_path:* (string) Extra path relative to the `Server.vardir` directory. (optional)

**Returns**:

string

#### Server:assert_follows_upstream (server_id)

Assert that the server follows the source node with the given ID. Meaning that it replicates from the remote node normally, and has already joined and subscribed.

**Parameters:**

- *server_id:* (**number**) Server ID.

#### Server:call (fn_name[, args[, options]])

Call remote function on the server by name.

This is a shortcut for `server.net_box:call()` .

**Parameters:**

- *fn_name:* (string)
- *args:* (**tab**) (optional)
- *options:* (**tab**) (optional)

#### Server:connect_net_box ()

Establish `net.box` connection. It's available in `net_box` field.

#### Server:copy_datadir ()

Copy contents of the data directory into the server's working directory. Invoked on the server's start.

#### Server:drop ()

Stop the server and save its artifacts if the test fails. This function should be used only at the end of the test (`after_test`, `after_each`, `after_all` hooks) to terminate the server process. Besides process termination, it saves the contents of the server working directory to the `<vardir>/artifacts` directory for further analysis if the test fails.

#### Server:eval (code[, args[, options]])

Evaluate Lua code on the server.

This is a shortcut for `server.net_box:eval()` .

**Parameters:**

- *code:* (string)
- *args:* (**tab**) (optional)
- *options:* (**tab**) (optional)

#### Server:exec (fn[, args[, options]])

Run given function on the server.

Much like `Server:eval` , but takes a function instead of a string. The executed function must have no upvalues (closures). Though it may use global functions and modules (like `box` , `os` , etc.)

**Parameters:**

- *fn:* (**function**)
- *args:* (**tab**) (optional)
- *options:* (**tab**) (optional)

#### Usage:

```
local vclock = server:exec(function()
    return box.info.vclock
end)

local sum = server:exec(function(a, b)
    return a + b
```

(continues on next page)

(продолжение с предыдущей страницы)

```
end, {1, 2})
-- sum == 3

local t = require('luatest')
server:exec(function()
    -- luatest is available via `t` upvalue
    t.assert_equals(math.pi, 3)
end)
-- mytest.lua:12: expected: 3, actual: 3.1415926535898
```

### Server:get_box_cfg ()

A simple wrapper around the `Server:exec()` method to get the `box.cfg` value from the server.

**Returns**:

table

### Server:get_downstream_vclock (server_id)

Get vclock acknowledged by another node to the current server.

**Parameters:**

- *server_id:* (**number**) Server ID.

**Returns**:

table

### Server:get_election_term ()

Get the election term as seen by the server.

**Returns**:

number

### Server:get_instance_id ()

Get ID of the server instance.

**Returns**:

number

### Server:get_instance_uuid ()

Get UUID of the server instance.

**Returns**:

string

### Server:get_synchro_queue_term ()

Get the synchro term as seen by the server.

**Returns**:

number

### Server:get_vclock ()

Get the server's own vclock, including the local component.

**Returns**:

table

### Server:grep_log (pattern[, bytes_num[, opts]])

Search a string pattern in the server's log file. If the server has crashed, `opts.filename` is required.

**Parameters:**

- *pattern:* (string) String pattern to search in the server's log file.
- *bytes_num:* (**number**) Number of bytes to read from the server's log file. (optional)
- *opts:*
  - *reset:* (**bool**) Reset the result when `Tarantool %d+.%d+.%d+-.*%d+-g.*` pattern is found, which means that the server was restarted.Defaults to `true` . (optional)
  - *filename:* (string) Path to the server's log file.Defaults to `box.cfg.log` . (optional)

**Returns**:

string|nil

### Server:http_request (method, path[, options])

Perform HTTP request.

**Parameters:**

- *method:* (string)
- *path:* (string)
- *options:*
  - *body:* (string) request body (optional)
  - *json:* data to encode as JSON into request body (optional)

- *http:* (**tab**) other options for HTTP-client (optional)
- *raise:* (**bool**) raise error when status is not in 200..299. Default to true. (optional)

**Returns**:

response object from HTTP client with helper methods.

**Raises:**

HTTPRequest error when response status is not 200.

**See also:**

- luatest.http_response

### Server:make_socketdir ()

Make directory for the server's Unix socket. Invoked on the server's start.

### Server:make_workdir ()

Make the server's working directory. Invoked on the server's start.

### Server:new ([object[, extra]])

Build a server object.

**Parameters:**

- *object:* Table with the entries listed below. (optional)
  - *command:* (string) Executable path to run a server process with.Defaults to the internal `server_instance.lua` script. If a custom pathis provided, it should correctly process all env variables listed belowto make constructor parameters work. (optional)
  - *args:* (**tab**) Arbitrary args to run `object.command` with. (optional)
  - *env:* (**tab**) Pass the given env variables into the server process. (optional)
  - *chdir:* (string) Change to the given directory before runningthe server process. (optional)
  - *alias:* (string) Alias for the new server and the value of the.. code-block:: lua `TARANTOOL_ALIAS` env variable which is passed into the server process.Defaults to „server“. (optional)
  - *workdir:* (string) Working directory for the new server and thevalue of the `TARANTOOL_WORKDIR` env variable which is passed into theserver process. The directory path will be created on the server start.Defaults to `<vardir>/<alias>-<random id>`. (optional)
  - *datadir:* (string) Directory path whose contents will be recursivelycopied into `object.workdir` on the server start. (optional)
  - *http_port:* (**number**) Port for HTTP connection to the new server andthe value of the `TARANTOOL_HTTP_PORT` env variable which is passed intothe server process.Not supported in the default `server_instance.lua` script. (optional)
  - *net_box_port:* (**number**) Port for the `net.box` connection to the newserver and the value of the `TARANTOOL_LISTEN` env variable which is passedinto the server process. (optional)

  - *net_box_uri:* (string) URI for the `net.box` connection to the newserver and the value of the `TARANTOOL_LISTEN` env variable which is passedinto the server process. If it is a Unix socket, the corresponding socketdirectory path will be created on the server start. (optional)
  - *net_box_credentials:* (**tab**) Override the default credentials for the.. code-block:: lua `net.box` connection to the new server. (optional)
  - *box_cfg:* (**tab**) Extra options for `box.cfg()` and the value of the.. code-block:: lua `TARANTOOL_BOX_CFG` env variable which is passed into the server process. (optional)
- *extra:* (**tab**) Table with extra properties for the server object. (optional)

**Returns**:

table

#### Server:play_wal_until_synchro_queue_is_busy ()

Play WAL until the synchro queue becomes busy. WAL records go one by one. The function is needed, because during `box.ctl.promote()` it is not known for sure which WAL record is PROMOTE - first, second, third? Even if known, it might change in the future. WAL delay should already be started before the function is called.

#### Server:restart ([params[, opts]])

Restart the server with the given parameters. Optionally waits until the server is ready.

**Parameters:**

- *params:* (**tab**) Parameters to restart the server with.Like `command` , `args` , `env` , etc. (optional)
- *opts:*
  - *wait_until_ready:* (**bool**) Wait until the server is ready.Defaults to `true` unless a custom executable path was provided whilebuilding the server object. (optional)

**See also:**

- luatest.server.Server:new

#### Server:start ([opts])

Start a server. Optionally waits until the server is ready.

**Parameters:**

- *opts:*
  - *wait_until_ready:* (**bool**) Wait until the server is ready.Defaults to `true` unless a custom executable was provided while buildingthe server object. (optional)

### Server:stop ()

Stop the server. Waits until the server process is terminated.

### Server:update_box_cfg (cfg)

A simple wrapper around the `Server:exec()` method to update the `box.cfg` value on the server.

**Parameters:**

- *cfg:* (**tab**) Box configuration settings.

### Server:wait_for_downstream_to (server)

Wait for the given server to reach at least the same vclock as the local server. Not including the local component, of course.

**Parameters:**

- *server:* (**tab**) Server's object.

### Server:wait_for_election_leader ()

Wait for the server to become a **writable** election leader.

### Server:wait_for_election_state (state)

Wait for the server to enter the given election state. Note that if it becomes a leader, it does not mean it is already writable.

**Parameters:**

- *state:* (string) Election state to wait for.

### Server:wait_for_election_term (term)

Wait for the server to reach at least the given election term.

**Parameters:**

- *term:* (string) Election term to wait for.

### Server:wait_for_synchro_queue_term (term)

Wait for the server to reach at least the given synchro term.

**Parameters:**

- *term:* (**number**) Synchro queue term to wait for.

#### Server:wait_for_vclock (vclock)

Wait until the server's own vclock reaches at least the given value. Including the local component.

**Parameters:**

- *vclock:* (**tab**) Server's own vclock to reach.

#### Server:wait_for_vclock_of (other_server)

Wait for the server to reach at least the same vclock as the other server. Not including the local component, of course.

**Parameters:**

- *other_server:* (**tab**) Other server's object.

#### Server:wait_until_election_leader_found ()

Wait for the server to discover an election leader.

#### Server:wait_until_ready ()

Wait until the server is ready after the start. A server is considered ready when its `_G.ready` variable becomes `true` .

### Class *luatest.replica_set*

Class to manage groups of Tarantool instances with the same data set.

#### Functions

Methods

#### ReplicaSet:add_server (server)

Add the server object to the replica set. The added server object should be built via the `ReplicaSet:build_server` function.

**Parameters:**

- *server:* (**tab**) Server object to be added to the replica set.

### ReplicaSet:build_and_add_server ([config])

Build a server object and add it to the replica set.

**Parameters:**

- *config:* (**tab**) Configuration for the new server. (optional)

**Returns**:

table

**See also:**

- luatest.server.Server:new

### ReplicaSet:build_server ([config])

Build a server object for the replica set.

**Parameters:**

- *config:* (**tab**) Configuration for the new server. (optional)

**Returns**:

table

**See also:**

- luatest.server.Server:new

### ReplicaSet:delete_server (alias)

Delete the server object from the replica set by the given server alias.

**Parameters:**

- *alias:* (string) Server alias.

### ReplicaSet:drop ()

Stop all servers in the replica set and save their artifacts if the test fails. This function should be used only at the end of the test (`after_test`, `after_each`, `after_all` hooks) to terminate all server processes in the replica set. Besides process termination, it saves the contents of each server working directory to the `<vardir>/artifacts` directory for further analysis if the test fails.

### ReplicaSet:get_leader ()

Get a server which is a writable node in the replica set.

**Returns**:

table

### ReplicaSet:get_server (alias)

Get the server object from the replica set by the given server alias.

**Parameters:**

- *alias:* (string) Server alias.

**Returns**:

table|nil

### ReplicaSet:new ([object])

Build a replica set object.

**Parameters:**

- *object:* Table with the entries listed below. (optional)
  - *servers:* (**tab**) List of server configurations to build serverobjects from and add them to the new replica set. See an example below. (optional)

**Returns**:

table

**See also:**

- luatest.server.Server:new

### Usage:

```
local ReplicaSet = require('luatest.replica_set')
local Server = require('luatest.server')
local box_cfg = {
    replication_timeout = 0.1,
    replication_connect_timeout = 10,
    replication_sync_lag = 0.01,
    replication_connect_quorum = 3,
    replication = {
        Server.build_listen_uri('replica1'),
        Server.build_listen_uri('replica2'),
        Server.build_listen_uri('replica3'),
    },
}
local replica_set = ReplicaSet:new({
    servers = {
        {alias = 'replica1', box_cfg = box_cfg},
        {alias = 'replica2', box_cfg = box_cfg},
        {alias = 'replica3', box_cfg = box_cfg},
    }
})
replica_set:start()
replica_set:wait_for_fullmesh()
```

#### ReplicaSet:start ([opts])

Start all servers in the replica set. Optionally waits until all servers are ready.

**Parameters:**

- *opts:* Table with the entries listed below. (optional)
  - *wait_until_ready:* (**bool**) Wait until all servers are ready.Defaults to `true` . (optional)

#### ReplicaSet:stop ()

Stop all servers in the replica set.

#### ReplicaSet:wait_for_fullmesh ([opts])

Wait until every node is connected to every other node in the replica set.

**Parameters:**

- *opts:* Table with the entries listed below. (optional)
  - *timeout:* (**number**) Timeout in seconds to wait for full mesh.Defaults to 60. (optional)
  - *delay:* (**number**) Delay in seconds between attempts to check full mesh.Defaults to 0.1. (optional)

### 9.6.3 Модуль vshard

В модуле `vshard` реализована функция продвинутого шардинга (сегментирования), которая основывается на понятии *виртуального сегмента* и позволяет осуществлять горизонтальное масштабирование в Tarantool.

To learn how sharding works in Tarantool, refer to the *Sharding* page.

Стоит также обратиться к *руководству по быстрому запуску*. Ниже приводятся справочники по модулю `vshard`:

#### Справочник по настройке

---

**Примечание:** Starting with the 3.0 version, the recommended way of configuring Tarantool is using a *configuration file*. Configuring Tarantool in code is considered a legacy approach.

---

#### Базовые параметры

- *sharding*
- *weights*
- *shard_index*
- *bucket_count*
- *collect_bucket_garbage_interval*

- *collect_lua_garbage*
- *sync_timeout*
- *rebalancer_disbalance_threshold*
- *rebalancer_max_receiving*
- *rebalancer_max_sending*
- *discovery_mode*
- *sched_move_quota*
- *sched_ref_quota*

`sharding`
: Поле, которое определяет логическую топологию сегментированного кластера Tarantool.

  Тип: таблица
  По умолчанию: false (ложь)
  Динамический: да

`weights`
: Поле, которое определяет конфигурацию относительного веса для каждой пары зон в наборе реплик. См. раздел *Вес реплики*.

  Тип: таблица
  По умолчанию: false (ложь)
  Динамический: да

`shard_index`
: Название или id TREE-индекса по *идентификатору сегмента*. Спейсы без этого индекса не задействованы в шардированном кластере Tarantool и при необходимости могут быть использованы как обычные спейсы. Необходимо указать первую часть индекса, остальные части являются необязательными.

  Тип: непустая строка или неотрицательное целое число
  По умолчанию: «bucket_id» (идентификатор сегмента)
  Динамический: нет

`bucket_count`
: Общее число сегментов в кластере.

  Это число должно быть на несколько порядков больше, чем потенциальное число узлов кластера, учитывая потенциальное масштабирование в обозримом будущем.

  **Пример:**

  Если предполагаемое количество узлов равно M, тогда набор данных должен быть разделен на 100M или даже 1000M сегментов, в зависимости от запланированного масштабирования. Это число, безусловно, больше потенциального числа узлов кластера в проектируемой системе.

Следует помнить, что слишком большое число сегментов может привести к необходимости выделять больше памяти для хранения информации о маршрутизации. С другой стороны, недостаточное число сегментов может привести к снижению степени детализации при балансировке.

Тип: число

По умолчанию: 3000

Динамический: нет

`collect_bucket_garbage_interval`

**Deprecated since:** 0.1.17.

Интервал между действиями сборщика мусора в секундах.

Тип: число

По умолчанию: 0.5

Динамический: да

`collect_lua_garbage`

**Deprecated since:** 0.1.20.

Если задано значение true (правда), периодически вызывается Lua-функция `collectgarbage()`.

Тип: логический

По умолчанию: нет

Динамический: да

`sync_timeout`

Время ожидания синхронизации старого мастера с репликами перед сменой мастера. Используется при переключении мастера или при вызове функции `sync()` вручную.

Тип: число

По умолчанию: 1

Динамический: да

`rebalancer_disbalance_threshold`

A maximum bucket disbalance threshold, in percent. The disbalance is calculated for each replica set using the following formula:

```
|эталонное_число_сегментов - фактическое_число_сегментов| / эталонное_число_
↪сегментов * 100
```

Тип: число

По умолчанию: 1

Динамический: да

`rebalancer_max_receiving`
: Максимальное количество сегментов, которые может получить параллельно один набор реплик. Это число должно быть ограничено, так как при добавлении нового набора реплик в кластер балансировщик отправляет очень большое количество сегментов из существующих наборов реплик в новый набор реплик. Это создает большую нагрузку на новый набор реплик.

  **Пример:**

  Предположим, `rebalancer_max_receiving` = 100, число сегментов в `bucket_count` = 1000. Есть 3 набора реплик с 333, 333 и 334 сегментами соответственно. При добавлении нового набора реплик `эталонное_число_сегментов` становится равным 250. Вместо того, чтобы сразу получить все 250 сегментов, новый набор реплик получит последовательно 100, 100 и 50 сегментов.

  Тип: число

  По умолчанию: 100

  Динамический: да

`rebalancer_max_sending`
: Степень параллельности для *параллельной балансировки*.

  Используется только для хранилищ, для роутеров игнорируется.

  Максимальное значение: 15.

  Тип: число

  По умолчанию: 1

  Динамический: да

`discovery_mode`
: Режим работы файбера обнаружения сегментов: `on`/`off`/`once`. *Подробнее*.

  Тип: строка

  По умолчанию: «on»

  Динамический: да

`sched_move_quota`
: A scheduler's bucket move quota used by the rebalancer.

  `sched_move_quota` defines how many bucket moves can be done in a row if there are pending storage refs. Then, bucket moves are blocked and a router continues making map-reduce requests.

  See also: *sched_ref_quota*.

  Тип: число

  По умолчанию: 1

  Динамический: да

`sched_ref_quota`
: A scheduler's storage ref quota used by a router's map-reduce API. For example, the *vshard.router.map_callrw()* function implements consistent map-reduce over the entire cluster.

  `sched_ref_quota` defines how many storage refs, therefore map-reduce requests, can be executed on the storage in a row if there are pending bucket moves. Then, storage refs are blocked and the rebalancer continues bucket moves.

  See also: *sched_move_quota*.

  Тип: число
  Default: 300
  Динамический: да

### Параметры набора реплик

- *uuid*
- *weight*
- *master*

`uuid`
: Уникальный идентификатор набора реплик.

  Тип:
  По умолчанию:
  Динамическое:

`weight`
: Вес набора реплик. Для получения подробной информации см. раздел *Вес набора реплик*.

  Тип:
  По умолчанию: 1
  Динамическое:

`master`
: Turns on automated master discovery in a replica set if set to `auto`. Applicable only to the configuration of a router; the storage configuration ignores this parameter.

  The parameter should be specified per replica set. The configuration is not compatible with a manual master selection.

  **Examples**

  Correct configuration:

```
config = {
    sharding = {
        <replicaset uuid> = {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            master = 'auto',
            replicas = {...},
        },
        ...
    },
    ...
}
```

Incorrect configuration:

```
config = {
    sharding = {
        <replicaset uuid> = {
            master = 'auto',
            replicas = {
                <replica uuid1> = {
                    master = true,
                    ...
                },
                <replica uuid2> = {
                    master = false,
                    ...
                },
            },
        },
        ...
    },
    ...
}
```

If the configuration is incorrect, it is not applied, and the `vshard.router.cfg()` call throws an error.

If the `master` parameter is set to `auto` for some replica sets, the router goes to these replica sets, discovers the master in each of them, and periodically checks if the master instance still has its master status. When the master in the replica set stops being a master, the router goes around all the nodes of the replica set to find out which one is the new master.

Without this setting, the router cannot detect master nodes in the configured replica sets on its own. It relies only on how they are specified in the configuration. This becomes a problem when the master changes, and the change is not delivered to the router's configuration: for instance, in case the router doesn't rely on a central configuration provider or the provider cannot deliver a new configuration due to some reason.

Тип: строка

Default: `nil`

Динамический: да

### API Reference

This section represents public and internal API for the *router* and the *storage*.

### Router API

| Subsection | Methods |
|---|---|
| *Router public API* | • *vshard.router.bootstrap()*<br>• *vshard.router.cfg(cfg)*<br>• *vshard.router.new(name, cfg)*<br>• *vshard.router.call(bucket_id, mode, function_name, {argument_list}, {options})*<br>• *vshard.router.callro(bucket_id, function_name, {argument_list}, {options})*<br>• *vshard.router.callrw(bucket_id, function_name, {argument_list}, {options})*<br>• *vshard.router.callre(bucket_id, function_name, {argument_list}, {options})*<br>• *vshard.router.callbro(bucket_id, function_name, {argument_list}, {options})*<br>• *vshard.router.callbre(bucket_id, function_name, {argument_list}, {options})*<br>• *vshard.router.map_callrw(function_name, {argument_list}, {options})*<br>• *vshard.router.route(bucket_id)*<br>• *vshard.router.routeall()*<br>• *vshard.router.bucket_id_strcrc32(key)*<br>• *vshard.router.bucket_id_mpcrc32(key)*<br>• *vshard.router.bucket_count()*<br>• *vshard.router.sync(timeout)*<br>• *vshard.router.discovery_wakeup()*<br>• *vshard.router.discovery_set()*<br>• *vshard.router.info({options})*<br>• *vshard.router.buckets_info()*<br>• *vshard.router.enable()*<br>• *vshard.router.disable()*<br>• *replicaset_object:call()*<br>• *replicaset_object:callro()*<br>• *replicaset_object:callrw()*<br>• *replicaset_object:callre()* |
| *Router internal API* | • *vshard.router.bucket_discovery(bucket_id)* |

### Router public API

`vshard.router.bootstrap()`
Perform the initial cluster bootstrap and distribute all buckets across the replica sets.

**Параметры**

- `timeout` – a number of seconds before ending a bootstrap attempt as unsuccessful. Recreate the cluster in case of bootstrap timeout.
- `if_not_bootstrapped` – by default is set to `false` that means raise an error, when the cluster is already bootstrapped. `True` means consider an already bootstrapped cluster a success.

**Example:**

```
vshard.router.bootstrap({timeout = 4, if_not_bootstrapped = true})
```

---

**Примечание:** To detect whether a cluster is bootstrapped, `vshard` looks for at least one bucket in the whole cluster. If the cluster was bootstrapped only partially (for example, due to an error during the first bootstrap), then it will be considered a bootstrapped cluster on a next bootstrap call with `if_not_bootstrapped`. So this is still a bad practice. Avoid calling `bootstrap()` multiple times.

---

`vshard.router.cfg(`*cfg*`)`
Configure the database and start sharding for the specified `router` instance. See the sample configuration.

**Параметры**

- `cfg` – a configuration table

`vshard.router.new(`*name*, *cfg*`)`
Create a new router instance. `vshard` supports multiple routers in a single Tarantool instance. Each router can be connected to any `vshard` cluster, and multiple routers can be connected to the same cluster.

A router created via `vshard.router.new()` works in the same way as a static router, but the method name is preceded by a colon (`vshard.router:method_name(...)`), while for a static router the method name is preceded by a period (`vshard.router.method_name(...)`).

A static router can be obtained via the `vshard.router.static()` method and then used like a router created via the `vshard.router.new()` method.

---

**Примечание:** `box.cfg` is shared among all the routers of a single instance.

---

**Параметры**

- `name` – a router instance name. This name is used as a prefix in logs of the router and must be unique within the instance
- `cfg` – a configuration table. See the sample configuration.

**Return** a router instance, if created successfully; otherwise, nil and an error object

`vshard.router.call(`*bucket_id*, *mode*, *function_name*, *{argument_list}*, *{options}*`)`
Call the function identified by function-name on the shard storing the bucket identified by bucket_id. See the *Processing requests* section for details on function operation.

**Параметры**

- `bucket_id` – a bucket identifier
- `mode` – either a string = „read“|“write“, or a map with mode=“read“|“write“ and/or prefer_replica=true|false and/or balance=true|false.
- `function_name` – a function to execute
- `argument_list` – an array of the function's arguments
- `options` –
  - `timeout` — a request timeout, in seconds. If the `router` cannot identify a shard with the specified `bucket_id`, it will retry until the timeout is reached.
  - other *net.box options*, such as `is_async`, `buffer`, `on_push` are also supported.

The mode parameter has two possible forms: a string or a map. Examples of the string form are: `'read'`, `'write'`. Examples of the map form are: `{mode='read'}`, `{mode='write'}`, `{mode='read', prefer_replica=true}`, `{mode='read', balance=true}`, `{mode='read', prefer_replica=true, balance=true}`.

If `'write'` is specified then the target is the master.

If `prefer_replica=true` is specified then the preferred target is one of the replicas, but the target is the master if there is no conveniently available replica.

It may be good to specify prefer_replica=true for functions which are expensive in terms of resource use, to avoid slowing down the master.

If `balance=true` then there is load balancing—reads are distributed over all the nodes in the replica set in round-robin fashion, with a preference for replicas if prefer_replica=true is also set.

**Return** The original return value of the executed function, or `nil` and error object. The error object has a type attribute equal to `ShardingError` or one of the regular Tarantool errors (`ClientError`, `OutOfMemory`, `SocketError`, etc.).

`ShardingError` is returned on errors specific for sharding: the master is missing, wrong bucket id, etc. It has an attribute code containing one of the values from the `vshard.error.code.*` LUA table, an optional attribute containing a message with the human-readable error description, and other attributes specific for the error code.

**Examples:**

To call `customer_add` function from `vshard/example`, say:

```
vshard.router.call(100,
                   'write',
                   'customer_add',
                   {{customer_id = 2, bucket_id = 100, name = 'name2', accounts = {}
↪}},
                   {timeout = 5})
-- or, the same thing but with a map for the second argument
vshard.router.call(100,
                   {mode='write'},
                   'customer_add',
                   {{customer_id = 2, bucket_id = 100, name = 'name2', accounts = {}
↪}},
                   {timeout = 5})
```

vshard.router.callro(*bucket_id*, *function_name*, *{argument_list}*, *{options}*)
Call the function identified by function-name on the shard storing the bucket identified by bucket_id, in read-only mode (similar to calling vshard.router.call with mode="read"). See the *Processing requests* section for details on function operation.

**Параметры**

- `bucket_id` – a bucket identifier
- `function_name` – a function to execute
- `argument_list` – an array of the function's arguments
- `options` –
  - `timeout` — a request timeout, in seconds.If the `router` cannot identify a shard with the specified `bucket_id`, it will retry until the timeout is reached.
  - other *net.box options*, such as `is_async`, `buffer`, `on_push` are also supported.

**Return**

The original return value of the executed function, or `nil` and error object. The error object has a type attribute equal to `ShardingError` or one of the regular Tarantool errors (`ClientError`, `OutOfMemory`, `SocketError`, etc.).

`ShardingError` is returned on errors specific for sharding: the replica set is not available, the master is missing, wrong bucket id, etc. It has an attribute code containing one of the values from the `vshard.error.code.*` LUA table, an optional attribute containing a message with the human-readable error description, and other attributes specific for this error code.

vshard.router.callrw(*bucket_id*, *function_name*, *{argument_list}*, *{options}*)
Call the function identified by function-name on the shard storing the bucket identified by bucket_id, in read-write mode (similar to calling vshard.router.call with mode="write"). See the *Processing requests* section for details on function operation.

**Параметры**

- `bucket_id` – a bucket identifier
- `function_name` – a function to execute
- `argument_list` – an array of the function's arguments
- `options` –
  - `timeout` — a request timeout, in seconds. If the `router` cannot identify a shard with the specified `bucket_id`, it will retry until the timeout is reached.
  - other *net.box options*, such as `is_async`, `buffer`, `on_push` are also supported.

**Return**

The original return value of the executed function, or `nil` and error object. The error object has a type attribute equal to `ShardingError` or one of the regular Tarantool errors (`ClientError`, `OutOfMemory`, `SocketError`, etc.).

`ShardingError` is returned on errors specific for sharding: the replica set is not available, the master is missing, wrong bucket id, etc. It has an attribute code containing one of the values from the `vshard.error.code.*` LUA table, an optional attribute containing a message with the human-readable error description, and other attributes specific for this error code.

vshard.router.callre(*bucket_id*, *function_name*, *{argument_list}*, *{options}*)
Call the function identified by function-name on the shard storing the bucket identified by bucket_id, in read-only mode (similar to calling `vshard.router.call` with `mode='read'`), with preference for a

replica rather than a master (similar to calling `vshard.router.call` with `prefer_replica = true`). See the *Processing requests* section for details on function operation.

**Параметры**

- `bucket_id` – a bucket identifier
- `function_name` – a function to execute
- `argument_list` – an array of the function's arguments
- `options` –
  - `timeout` — a request timeout, in seconds. If the `router` cannot identify a shard with the specified `bucket_id`, it will retry until the timeout is reached.
  - other *net.box options*, such as `is_async`, `buffer`, `on_push` are also supported.

**Return**

The original return value of the executed function, or `nil` and error object. The error object has a type attribute equal to `ShardingError` or one of the regular Tarantool errors (`ClientError`, `OutOfMemory`, `SocketError`, etc.).

`ShardingError` is returned on errors specific for sharding: the replica set is not available, the master is missing, wrong bucket id, etc. It has an attribute code containing one of the values from the `vshard.error.code.*` LUA table, an optional attribute containing a message with the human-readable error description, and other attributes specific for this error code.

`vshard.router.callbro(`*bucket_id*, *function_name*, *{argument_list}*, *{options}*`)`
This has the same effect as *vshard.router.call()* with mode parameter = `{mode='read', balance=true}`.

`vshard.router.callbre(`*bucket_id*, *function_name*, *{argument_list}*, *{options}*`)`
This has the same effect as *vshard.router.call()* with mode parameter = `{mode='read', balance=true, prefer_replica=true}`.

`vshard.router.map_callrw(`*function_name*, *{argument_list}*, *{options}*`)`
The function implements consistent map-reduce over the entire cluster. Consistency means:

- All the data was accessible.
- The data was not migrated between physical storages during the map requests execution.

The function can be helpful if you need to access:

- all the data in the cluster
- a vast number of buckets scattered over the instances in case their individual *vshard.router.call()* takes up too much time.

The function is called on the master node of each replica set with the given arguments.

**Параметры**

- `function_name` – a function to call on the storages (masters of all replica sets)
- `argument_list` – an array of the function's arguments
- `options` –
  - `timeout` – a request timeout, in seconds. The timeout is for the entire `map_callrw()`, including all its stages.

– `return_raw` – the *net.box option* implemented in Tarantool since version 2.10.0. If set to `true`, `net.box` returns the response data wrapped in a *MessagePack object* instead of decoding it to Lua. For more details, see the **Return** section below.

---

**Важно:** Do not use a big timeout (longer than 1 minute, for instance). The router tries to block the bucket moves to another storage for the given timeout on all storages. On failure, the block remains for the entire timeout.

---

**Return**

- On success: a map with replica set UUIDs (keys) and results of the `function_name` (values).

```
{uuid1 = {res1}, uuid2 = {res2}, ...}
```

If the function returns `nil` or `box.NULL` from one of the storages, it will not be present in the resulting map.

If the `return_raw` option is used, the result is a map of the following format: `{[replicaset_uuid] = msgpack.object}` where `msgpack.object` is an object that stores a MessagePack array with the results returned from the storage map function.

The option use case is the same as in using `net.box`: to avoid decoding of the call results into Lua. The option can be helpful if a router is used as a proxy and results received from a storage are big.

Example:

```
local res = vshard.router.map_callrw('my_func', args, {..., return_raw = true})

for replicaset_uuid, msgpack_value in pairs(res) do
    log.info('Replicaset %s returned %s', replicaset_uuid,
             msgpack_value:decode())
end
```

This is an illustration of the option usage. Normally, you don't need to use `return_raw` if you call the *decode()* function.

- On failure: `nil`, error object, and optional replica set UUID where the error occurred. UUID will not be returned if the error is not related to a particular replica set. For instance, the method fails if not all buckets were found, even if all replica sets were scanned successfully. Handling the result looks like this:

```
res, err, uuid = vshard.router.map_callrw(...)
if not res then
    -- Error.
    -- 'err' - error object. 'uuid' - optional UUID of replica set
    -- where the error happened.
    ...
else
    -- Success.
    for uuid, value in pairs(res) do
        ...
    end
end
```

If the `return_raw` option is used, the result on failure is the same as described above.

Map-Reduce in vshard can be divided into three stages: Ref, Map, and Reduce.

**Ref and Map**. `map_callrw()` combines both the Ref and the Map stages. The Ref stage ensures data consistency while executing the user's function (`function_name`) on all nodes. Keep in mind that consistency is incompatible with rebalancing (it breaks data consistency). Map-reduce and rebalancing are mutually exclusive, they compete for the cluster time. Any bucket move makes the sender and receiver nodes inconsistent, so it is impossible to call a function on them to access all the data without *vshard.storage.bucket_ref()*. It makes the Ref stage intricate, as it should work together with the rebalancer to ensure they do not block each other.

For this, the storage has a special scheduler for bucket moves and storage refs. Storage ref is a volatile counter defined on each instance. It is incremented when a map-reduce request comes and decremented when it ends. Storage ref pins the entire instance with all its buckets, not just a single bucket (like bucket ref).

The scheduler shares storage time between bucket moves and storage refs fairly. The distribution depends on how long and frequent the moves and refs are. It can be configured using the storage options *sched_move_quota* and *sched_ref_quota*. Keep in mind that the scheduler configuration may affect map-reduce requests if used during rebalancing.

During the Map stage, `map_callrw()` sends map requests one by one to many servers. On success, the function returns a map. The map is a set of «key—value» pairs. The keys are replica set UUIDs, and the values are the results of the user's function—`function_name`.

**Reduce**. The Reduce stage is not performed by vshard. It is what the user's code does with the results of `map_callrw()`.

---

**Примечание:** `map_callrw()` works only on masters. Therefore, you can't use it if at least one replica set has its master node down.

---

`vshard.router.route(`*bucket_id*`)`

Return the replica set object for the bucket with the specified bucket id value.

**Параметры**

- `bucket_id` – a bucket identifier

**Return** a replica set object

**Example:**

```
replicaset = vshard.router.route(123)
```

`vshard.router.routeall()`

Return all available replica set objects.

**Return** a map of the following type: `{UUID = replicaset}`

**Rtype** a map of replica set objects

**Example:**

```
function selectall()
    local resultset = {}
    shards, err = vshard.router.routeall()
    if err ~= nil then
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        error(err)
    end
    for uid, replica in pairs(shards) do
        local set = replica:callro('box.space.*space-name*:select', {{}, {limit=10}}
↪, {timeout=5})
        for _, item in ipairs(set) do
            table.insert(resultset, item)
        end
    end
    table.sort(resultset, function(a, b) return a[1] < b[1] end)
    return resultset
end
```

vshard.router.bucket_id(*key*)

**Deprecated**. Logs a warning when used because it is not consistent for cdata numbers.

In particular, it returns 3 different values for normal Lua numbers like 123, for unsigned long long cdata (like `123ULL`, or `ffi.cast('unsigned long long',123)`), and for signed long long cdata (like `123LL`, or `ffi.cast('long long', 123)`). And it is important.

```
vshard.router.bucket_id(123)
vshard.router.bucket_id(123LL)
vshard.router.bucket_id(123ULL)
```

For float and double cdata (`ffi.cast('float', number)`, `ffi.cast('double', number)`) these functions return different values even for the same numbers of the same floating point type. This is because `tostring()` on a floating point cdata number returns not the number, but a pointer at it. Different on each call.

`vshard.router.bucket_id_strcrc32()` behaves exactly the same, but does not log a warning. In case you need that behavior.

vshard.router.bucket_id_strcrc32(*key*)

Calculate the bucket id using a simple built-in hash function.

**Параметры**

- `key` – a hash key. This can be any Lua object (number, table, string).

**Return** a bucket identifier

**Rtype** number

**Example:**

```
tarantool> vshard.router.bucket_count()
---
- 3000
...

tarantool> vshard.router.bucket_id_strcrc32("18374927634039")
---
- 2032
...

tarantool> vshard.router.bucket_id_strcrc32(18374927634039)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
---
- 2032
...

tarantool> vshard.router.bucket_id_strcrc32("test")
---
- 1216
...

tarantool> vshard.router.bucket_id_strcrc32("other")
---
- 2284
...
```

**Примечание:** Remember that it is not safe. See details in *bucket_id()*

vshard.router.bucket_id_mpcrc32(*key*)

This function is safer than `bucket_id_strcrc32`. It takes a CRC32 from a MessagePack encoded value. That is, bucket id of integers does not depend on their Lua type. In case of a string key, it does not encode it into MessagePack, but takes a hash right from the string.

**Параметры**

- `key` – a hash key. This can be any Lua object (number, table, string).

**Return** a bucket identifier

**Rtype** number

However it still may return different values for not equal floating point types. That is, `ffi.cast('float', number)` may be reflected into a bucket id not equal to `ffi.cast('double', number)`. This can't be fixed, because a float value, even being casted to double, may have a garbage tail in its fraction.

Floating point keys should not be used to calculate a bucket id, usually.

Be very careful in case you store floating point types in a space. When data is returned from a space, it is cast to Lua number. And if that value had an empty fraction part, it will be treated as an integer by `bucket_id_mpcrc32()`. So you need to do explicit casts in such cases. Here is an example of the problem:

```
tarantool> s = box.schema.create_space('test', {format = {{'id', 'double'}}}); _ =␣
↪s:create_index('pk')
---
...

tarantool> inserted = ffi.cast('double', 1)
---
...

-- Value is stored as double
tarantool> s:replace({inserted})
---
- [1]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...

-- But when returned to Lua, stored as Lua number, not cdata.
tarantool> returned = s:get({inserted}).id
---
...

tarantool> type(returned), returned
---
- number
- 1
...

tarantool> vshard.router.bucket_id_mpcrc32(inserted)
---
- 1411
...
tarantool> vshard.router.bucket_id_mpcrc32(returned)
---
- 1614
...
```

`vshard.router.bucket_count()`
Return the total number of buckets specified in `vshard.router.cfg()`.

**Return** the total number of buckets

**Rtype** number

```
tarantool> vshard.router.bucket_count()
---
- 10000
...
```

`vshard.router.sync(`*timeout*`)`
Wait until the dataset is synchronized on replicas.

**Параметры**

- `timeout` – a timeout, in seconds

**Return** `true` if the dataset was synchronized successfully; or `nil` and `err` explaining why the dataset cannot be synchronized.

`vshard.router.discovery_wakeup()`
Force wakeup of the bucket discovery fiber.

`vshard.router.discovery_set(`*mode*`)`
Turn on/off the background discovery fiber used by the router to find buckets.

**Параметры**

- `mode` – working mode of a discovery fiber. There are three modes: `on`, `off` and `once`

When the mode is `on` (default), the discovery fiber works during all the lifetime of the router. Even after all buckets are discovered, it will still come to storages and download their buckets with some big period (DISCOVERY_IDLE_INTERVAL). This is useful if the bucket topology changes often and

the number of buckets is not big. The router will keep its route table up to date even when no requests are processed.

When the mode is `off`, discovery is disabled completely.

When the mode is `once`, discovery starts and finds the locations of all buckets, and then the discovery fiber is terminated. This is good for a large bucket count and for clusters, where rebalancing is rare.

The method is good to enable/disable discovery after the router is already started, but discovery is enabled by default. You may want to never enable it even for a short time—then specify the `discovery_mode` option in the *configuration*. It takes the same values as `vshard.router.discovery_set(`*`mode`*`)`.

You may decide to turn off discovery or make it `once` if you have many routers, or tons of buckets (hundreds of thousands and more), and you see that the discovery process consumes notable CPU % on routers and storages. In that case it may be wise to turn off the discovery when there is no rebalancing in the cluster. And turn it on for new routers, as well as for all routers when rebalancing is started.

`vshard.router.info(`*{options}*`)`

Return information about each instance. Since vshard v.0.1.22, the function also accepts options, which can be used to get additional information.

**Параметры**

- `options` –
  - `with_services` — a bool value. If set to `true`, the function returns information about the background services (such as discovery, master search, or failover) that are working on the current instance.

**Return**

Replica set parameters:

- replica set uuid
- master instance parameters
- replica instance parameters

Instance parameters:

- `uri`—URI of the instance
- `uuid`—UUID of the instance
- `status`—status of the instance (`available`, `unreachable`, `missing`)
- `network_timeout`—a timeout for the request. The value is updated automatically on each 10th successful request and each 2nd failed request.

Bucket parameters:

- `available_ro` – the number of buckets known to the `router` and available for read requests
- `available_rw` – the number of buckets known to the `router` and available for read and write requests
- `unreachable` – the number of buckets known to the `router` but unavailable for any requests
- `unknown` – the number of buckets whose replica sets are not known to the `router`

Service parameters:

- `name` – service name. Possible values: `discovery`, `failover`, `master_search`.
- `status` – service status. Possible values: `ok`, `error`.

- `error` – error message that appears on the `error` status.
- `activity` – service state. It shows what the service is currently doing (for example, `updating replicas`).
- `status_idx` – incrementing counter of the status changes. The `ok` status is updated on every successful iteration of the service. The `error` status is updated only when it is fixed.

**Example:**

```
tarantool> vshard.router.info()
---
- replicasets:
    ac522f65-aa94-4134-9f64-51ee384f1a54:
      replica: &0
        network_timeout: 0.5
        status: available
        uri: storage@127.0.0.1:3303
        uuid: 1e02ae8a-afc0-4e91-ba34-843a356b8ed7
      uuid: ac522f65-aa94-4134-9f64-51ee384f1a54
      master: *0
    cbf06940-0790-498b-948d-042b62cf3d29:
      replica: &1
        network_timeout: 0.5
        status: available
        uri: storage@127.0.0.1:3301
        uuid: 8a274925-a26d-47fc-9e1b-af88ce939412
      uuid: cbf06940-0790-498b-948d-042b62cf3d29
      master: *1
  bucket:
    unreachable: 0
    available_ro: 0
    unknown: 0
    available_rw: 3000
  status: 0
  alerts: []
...

tarantool> vshard.router.info({with_services = true})
---
<all info from vshard.router.info()>
  services:
    failover:
      status_idx: 2
      error:
      activity: idling
      name: failover
      status: ok
    discovery:
      status_idx: 2
      error: Error during discovery: TimedOut
      activity: idling
      name: discovery
      status: error
...
```

vshard.router.buckets_info()
Return information about each bucket. Since a bucket map can be huge, only the required range of buckets can be specified.

**Параметры**

- `offset` – the offset in a bucket map of the first bucket to show
- `limit` – the maximum number of buckets to show

**Return** a map of the following type: `{bucket_id = 'unknown'/replicaset_uuid}`

```
tarantool> vshard.router.buckets_info()
---
- - uuid: aaaaaaaa-0000-4000-a000-000000000000
    status: available_rw
  - uuid: aaaaaaaa-0000-4000-a000-000000000000
    status: available_rw
  - uuid: aaaaaaaa-0000-4000-a000-000000000000
    status: available_rw
  - uuid: bbbbbbbb-0000-4000-a000-000000000000
    status: available_rw
  - uuid: bbbbbbbb-0000-4000-a000-000000000000
    status: available_rw
  - uuid: bbbbbbbb-0000-4000-a000-000000000000
    status: available_rw
  - uuid: bbbbbbbb-0000-4000-a000-000000000000
    status: available_rw
...
```

vshard.router.enable()
Since vshard v.0.1.21. Manually allow access to the router API, revert *vshard.router.disable()*.

---

**Примечание:** `vshard.router.enable()` cannot be used for enabling a router API that was automatically disabled due to a running configuration process.

---

vshard.router.disable()
Since vshard v.0.1.21. Manually restrict access to the router API. When the API is disabled, all its methods throw a Lua error, except *vshard.router.cfg()*, *vshard.router.new()*, *vshard.router.enable()* and *vshard.router.disable()*. The error object's name attribute is `ROUTER_IS_DISABLED`.

The router is enabled by default. However, it is automatically and forcefully disabled until the configuration is finished, as accessing the router's methods at that time is not safe.

Manual disabling can be used, for example, if some preparatory work needs to be done after calling *vshard.router.cfg()* but before the router's methods are available. It will look like this:

```
vshard.router.disable()
vshard.router.cfg(...)
-- Some preparatory work here ...
vshard.router.enable()
-- vshard.router's methods are available now
```

object replicaset_object

`replicaset_object:call`(*function_name*, *{argument_list}*, *{options}*)
Call a function on a nearest available master (distances are defined using `replica.zone` and `cfg.weights` matrix) with specified arguments.

---

**Примечание:** The `replicaset_object:call` method is similar to `replicaset_object:callrw`.

---

**Параметры**

- `function_name` – function to execute
- `argument_list` – array of the function's arguments
- `options` –
  - `timeout` — a request timeout, in seconds. If the `router` cannot identify a shard with the specified `bucket_id`, it will retry until the timeout is reached.
  - other *net.box options*, such as `is_async`, `buffer`, `on_push` are also supported.

**Return**

- result of `function_name` on success
- nil, err otherwise

`replicaset_object:callrw`(*function_name*, *{argument_list}*, *{options}*)
Call a function on a nearest available master (distances are defined using `replica.zone` and `cfg.weights` matrix) with a specified arguments.

---

**Примечание:** The `replicaset_object:callrw` method is similar to `replicaset_object:call`.

---

**Параметры**

- `function_name` – function to execute
- `argument_list` – array of the function's arguments
- `options` –
  - `timeout` — a request timeout, in seconds. If the `router` cannot identify a shard with the specified `bucket_id`, it will retry until the timeout is reached.
  - other *net.box options*, such as `is_async`, `buffer`, `on_push` are also supported.

**Return**

- result of `function_name` on success
- nil, err otherwise

```
tarantool> local bucket = 1; return vshard.router.callrw(
         >     bucket,
         >     'box.space.actors:insert',
         >     {{
         >         1, bucket, 'Renata Litvinova',
         >         {theatre="Moscow Art Theatre"}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
> }},
> {timeout=5}
> )
```

`replicaset_object:callro(`*function_name*, *{argument_list}*, *{options}*`)`
Call a function on the nearest available replica (distances are defined using `replica.zone` and `cfg.weights` matrix) with specified arguments. It is recommended to use `replicaset_object:callro()` for calling only read-only functions, as the called functions can be executed not only on a master, but also on replicas.

**Параметры**

- `function_name` – function to execute
- `argument_list` – array of the function's arguments
- `options` –
  - `timeout` — a request timeout, in seconds. If the `router` cannot identify a shard with the specified `bucket_id`, it will retry until the timeout is reached.
  - other *net.box options*, such as `is_async`, `buffer`, `on_push` are also supported.

**Return**

- result of `function_name` on success
- nil, err otherwise

`replicaset:callre(`*function_name*, *{argument_list}*, *{options}*`)`
Call a function on the nearest available replica (distances are defined using `replica.zone` and `cfg.weights` matrix) with specified arguments, with preference for a replica rather than a master (similar to calling `vshard.router.call` with `prefer_replica = true`). It is recommended to use `replicaset_object:callre()` for calling only read-only functions, as the called function can be executed not only on a master, but also on replicas.

**Параметры**

- `function_name` – function to execute
- `argument_list` – array of the function's arguments
- `options` –
  - `timeout` — a request timeout, in seconds. If the `router` cannot identify a shard with the specified `bucket_id`, it will retry until the timeout is reached.
  - other *net.box options*, such as `is_async`, `buffer`, `on_push` are also supported.

**Return**

- result of `function_name` on success
- nil, err otherwise

`vshard.router.master_search_wakeup()`
*Automated master discovery* works in its own fiber on a router, which is activated only if at least one replica set is configured to look for the master (the `master` parameter is set to `auto`). The fiber wakes up within a certain period. But it is possible to wake it up on demand by using this function.

Manual fiber wakeup can help speed up tests for master change. Another use case is performing some actions with a router in the router console.

The function does nothing if master search is not *configured* for any replica set.

**Return** none

### Router internal API

vshard.router.bucket_discovery(*bucket_id*)

Search for the bucket in the whole cluster. If the bucket is not found, it is likely that it does not exist. The bucket might also be moved during rebalancing and currently is in the RECEIVING state.

**Параметры**

- `bucket_id` – a bucket identifier

### Storage API

| | |
|---|---|
| *Storage public API* | • *vshard.storage.cfg(cfg, instance_uuid)*<br>• *vshard.storage.info({options})*<br>• *vshard.storage.call(bucket_id, mode, function_name, {argument_list})*<br>• *vshard.storage.sync(timeout)*<br>• *vshard.storage.bucket_pin(bucket_id)*<br>• *vshard.storage.bucket_unpin(bucket_id)*<br>• *vshard.storage.bucket_ref(bucket_id, mode)*<br>• *vshard.storage.bucket_refro(bucket_id)*<br>• *vshard.storage.bucket_refrw(bucket_id)*<br>• *vshard.storage.bucket_unref(bucket_id, mode)*<br>• *vshard.storage.bucket_unrefro(bucket_id)*<br>• *vshard.storage.bucket_unrefrw(bucket_id)*<br>• *vshard.storage.find_garbage_bucket(bucket_index, control)*<br>• *vshard.storage.rebalancer_disable()*<br>• *vshard.storage.rebalancer_enable()*<br>• *vshard.storage.is_locked()*<br>• *vshard.storage.rebalancing_is_in_progress()*<br>• *vshard.storage.buckets_info()*<br>• *vshard.storage.buckets_count()*<br>• *vshard.storage.sharded_spaces()*<br>• *vshard.storage.on_bucket_event()* |
| *Storage internal API* | • *vshard.storage.bucket_stat(bucket_id)*<br>• *vshard.storage.bucket_recv(bucket_id, from, data)*<br>• *vshard.storage.bucket_delete_garbage(bucket_id)*<br>• *vshard.storage.bucket_collect(bucket_id)*<br>• *vshard.storage.bucket_force_create(first_bucket_id, count)*<br>• *vshard.storage.bucket_force_drop(bucket_id, to)*<br>• *vshard.storage.bucket_send(bucket_id, to)*<br>• *vshard.storage.buckets_discovery()*<br>• *vshard.storage.rebalancer_request_state()* |

### Storage public API

vshard.storage.cfg(*cfg*, *instance_uuid*)
Configure the database and start sharding for the specified `storage` instance.

**Параметры**

- `cfg` – a `storage` configuration
- `instance_uuid` – UUID of the instance

vshard.storage.info(*{options}*)
Return information about the storage instance. Since vshard v.0.1.22, the function also accepts options, which can be used to get additional information.

**Параметры**

- `options` –
  - `with_services` — a bool value. If set to `true`, the function returns information about the background services (such as garbage collector, rebalancer, recovery, or applier of the routes) that are working on the current instance. See *vshard.router.info* for detailed reference.

**Example:**

```
tarantool> vshard.storage.info()
---
- replicasets:
    c862545d-d966-45ff-93ad-763dce4a9723:
      uuid: c862545d-d966-45ff-93ad-763dce4a9723
      master:
        uri: admin@localhost:3302
    1990be71-f06e-4d9a-bcf9-4514c4e0c889:
      uuid: 1990be71-f06e-4d9a-bcf9-4514c4e0c889
      master:
        uri: admin@localhost:3304
  bucket:
    receiving: 0
    active: 15000
    total: 15000
    garbage: 0
    pinned: 0
    sending: 0
  status: 0
  replication:
    status: master
  alerts: []
...
```

vshard.storage.call(*bucket_id*, *mode*, *function_name*, *{argument_list}*)
Call the specified function on the current `storage` instance.

**Параметры**

- `bucket_id` – a bucket identifier
- `mode` – a type of the function: „read“ or „write“
- `function_name` – function to execute

- `argument_list` – array of the function's arguments

**Return**

The original return value of the executed function, or `nil` and error object.

`vshard.storage.sync(`*timeout*`)`
Wait until the dataset is synchronized on replicas.

**Параметры**

- `timeout` – a timeout, in seconds

**Return** `true` if the dataset was synchronized successfully; or `nil` and `err` explaining why the dataset cannot be synchronized.

`vshard.storage.bucket_pin(`*bucket_id*`)`
Pin a bucket to a replica set. A pinned bucket cannot be moved even if it breaks the cluster balance.

**Параметры**

- `bucket_id` – a bucket identifier

**Return** `true` if the bucket is pinned successfully; or `nil` and `err` explaining why the bucket cannot be pinned

`vshard.storage.bucket_unpin(`*bucket_id*`)`
Return a pinned bucket back into the active state.

**Параметры**

- `bucket_id` – a bucket identifier

**Return** `true` if the bucket is unpinned successfully; or `nil` and `err` explaining why the bucket cannot be unpinned

`vshard.storage.bucket_ref(`*bucket_id*, *mode*`)`
Create an RO or RW *ref*.

**Параметры**

- `bucket_id` – a bucket identifier
- `mode` – „read“ or „write“

**Return** `true` if the bucket ref is created successfully; or `nil` and `err` explaining why the ref cannot be created

`vshard.storage.bucket_refro(`*bucket_id*`)`
An alias for *vshard.storage.bucket_ref* in the RO mode.

**Параметры**

- `bucket_id` – a bucket identifier

**Return** `true` if the bucket ref is created successfully; or `nil` and `err` explaining why the ref cannot be created

`vshard.storage.bucket_refrw(`*bucket_id*`)`
An alias for *vshard.storage.bucket_ref* in the RW mode.

**Параметры**

- `bucket_id` – a bucket identifier

**Return** `true` if the bucket ref is created successfully; or `nil` and `err` explaining why the ref cannot be created

vshard.storage.bucket_unref(*bucket_id*, *mode*)
Remove a RO/RW *ref*.

**Параметры**

- `bucket_id` – a bucket identifier
- `mode` – „read“ or „write“

**Return** `true` if the bucket ref is removed successfully; or `nil` and `err` explaining why the ref cannot be removed

vshard.storage.bucket_unrefro(*bucket_id*)
An alias for *vshard.storage.bucket_unref* in the RO mode.

**Параметры**

- `bucket_id` – a bucket identifier

**Return** `true` if the bucket ref is removed successfully; or `nil` and `err` explaining why the ref cannot be removed

vshard.storage.bucket_unrefrw(*bucket_id*)
An alias for *vshard.storage.bucket_unref* in the RW mode.

**Параметры**

- `bucket_id` – a bucket identifier

**Return** `true` if the bucket ref is removed successfully; or `nil` and `err` explaining why the ref cannot be removed

vshard.storage.find_garbage_bucket(*bucket_index*, *control*)
Find a bucket which has data in a space but is not stored in a `_bucket` space; or is in a GARBAGE state.

**Параметры**

- `bucket_index` – index of a space with the part of a bucket id
- `control` – a garbage collector controller. If there is an increased buckets generation, then the search should be interrupted.

**Return** an identifier of the bucket in the garbage state, if found; otherwise, nil

vshard.storage.buckets_info()
Return information about each bucket located in storage. For example:

```
tarantool> vshard.storage.buckets_info(1)
---
- 1:
    status: active
    ref_rw: 1
    ref_ro: 1
    ro_lock: true
    rw_lock: true
    id: 1
```

vshard.storage.buckets_count()
Return the number of buckets located in storage.

vshard.storage.recovery_wakeup()
Immediately wake up a recovery fiber, if it exists.

`vshard.storage.rebalancing_is_in_progress()`
Return a flag indicating whether rebalancing is in progress. The result is true if the node is currently applying routes received from a rebalancer node in the special fiber.

`vshard.storage.is_locked()`
Return a flag indicating whether storage is invisible to the rebalancer.

`vshard.storage.rebalancer_disable()`
Disable rebalancing. A disabled rebalancer sleeps until it is enabled again with vshard.storage.rebalancer_enable().

`vshard.storage.rebalancer_enable()`
Enable rebalancing.

`vshard.storage.sharded_spaces()`
Show the spaces that are visible to rebalancer and garbage collector fibers.

```
tarantool> vshard.storage.sharded_spaces()
---
- 513:
    engine: memtx
    before_replace: 'function: 0x010e50e738'
    field_count: 0
    id: 513
    on_replace: 'function: 0x010e50e700'
    temporary: false
    index:
      0: &0
        unique: true
        parts:
        - type: number
          fieldno: 1
          is_nullable: false
        id: 0
        type: TREE
        name: primary
        space_id: 513
      1: &1
        unique: false
        parts:
        - type: number
          fieldno: 2
          is_nullable: false
        id: 1
        type: TREE
        name: bucket_id
        space_id: 513
      primary: *0
      bucket_id: *1
    is_local: false
    enabled: true
    name: actors
    ck_constraint: []
...
```

vshard.storage.on_bucket_event([*trigger-function*[, *old-trigger-function*]])

Since vshard v.0.1.22. Define a trigger for execution when the data from the user spaces is changed (deleted or inserted) due to the rebalancing process. The trigger is invoked each time the data batch changes.

**Параметры**

- `trigger-function` (`function`) – function which will become the trigger function.
- `old-trigger-function` (`function`) – existing trigger function which will be replaced by trigger-function.

**Return** nil or function pointer

The `trigger-function` can have up to three parameters:

- `event_type` (string) – in order to distinguish event, you can compare this argument with the supported event types, `bucket_data_recv_txn` and `bucket_data_gc_txn`.
- `bucket_id` (unsigned) – bucket id.
- `data` (table) – additional information about data change transaction. Currently it only includes an array of all spaces (`data.spaces`), affected by a transaction in which trigger-function is executed.

**Example:**

```
vshard.storage.on_bucket_event(function(event, bucket_id, data)
    if event == 'bucket_data_recv_txn' then
        -- Handle it.
        for idx, space in ipairs(data.spaces) do
            ...
        end
    elseif event == 'bucket_data_gc_txn' then
        -- Handle it.
        ...
    end
end)
```

---

**Примечание:** As everything executed inside triggers is already in a transaction, you shouldn't use transactions, yield-operations (*explicit* or not), changes to different space engines (see *rule #2*).

If the parameters are `(nil, old-trigger-function)`, then the old trigger is deleted. If both parameters are omitted, then the response is a list of existing trigger functions.

Details about trigger characteristics are in the *triggers* section.

---

### Storage internal API

vshard.storage.bucket_recv(*bucket_id*, *from*, *data*)

Receive a bucket identified by bucket id from a remote replica set.

**Параметры**

- `bucket_id` – a bucket identifier
- `from` – UUID of source replica set
- `data` – data logically stored in a bucket identified by bucket_id, in the same format as the return value from `bucket_collect()` <storage_api-bucket_collect>

vshard.storage.bucket_stat(*bucket_id*)
Return information about the bucket id:

```
tarantool> vshard.storage.bucket_stat(1)
---
- 0
- status: active
  id: 1
...
```

**Параметры**

- `bucket_id` – a bucket identifier

vshard.storage.bucket_delete_garbage(*bucket_id*)
Force garbage collection for the bucket identified by bucket_id in case the bucket was transferred to a different replica set.

**Параметры**

- `bucket_id` – a bucket identifier

vshard.storage.bucket_collect(*bucket_id*)
Collect all the data that is logically stored in the bucket identified by bucket_id:

```
tarantool> vshard.storage.bucket_collect(1)
---
- 0
- - - 514
    - - [10, 1, 1, 100, 'Account 10']
      - [11, 1, 1, 100, 'Account 11']
      - [12, 1, 1, 100, 'Account 12']
      - [50, 5, 1, 100, 'Account 50']
      - [51, 5, 1, 100, 'Account 51']
      - [52, 5, 1, 100, 'Account 52']
  - - 513
    - - [1, 1, 'Customer 1']
      - [5, 1, 'Customer 5']
...
```

**Параметры**

- `bucket_id` – a bucket identifier

vshard.storage.bucket_force_create(*first_bucket_id*, *count*)
Force creation of the buckets (single or multiple) on the current replica set. Use only for manual emergency recovery or for initial bootstrap.

**Параметры**

- `first_bucket_id` – an identifier of the first bucket in a range
- `count` – the number of buckets to insert (default = 1)

vshard.storage.bucket_force_drop(*bucket_id*)
Drop a bucket manually for tests or emergency cases.

**Параметры**

- `bucket_id` – a bucket identifier

`vshard.storage.bucket_send(`*bucket_id*, *to*`)`
Send a specified bucket from the current replica set to a remote replica set.

**Параметры**

- `bucket_id` – bucket identifier
- `to` – UUID of a remote replica set

`vshard.storage.rebalancer_request_state()`
Check all buckets of the host storage that have the SENT or ACTIVE state, return the number of active buckets.

**Return** the number of buckets in the active state, if found; otherwise, nil

`vshard.storage.buckets_discovery()`
Collect an array of active bucket identifiers for discovery.

### 9.6.4 Модули СУБД SQL

В данном разделе справочника рассматривается внедрение и использование двух уже созданных модулей: сторонние библиотеки СУБД SQL для MySQL и PostgreSQL.

Для вызова другой СУБД из Tarantool нужно: другая СУБД и Tarantool. Модуль, который соединяет другую СУБД может называться коннектором. В модуле есть библиотека общего пользования, которая может называться драйвером.

Tarantool предоставляет модули-коннекторы для СУБД вместе с менеджером модулей для Lua под названием LuaRocks.

Модули Tarantool позволяют подключаться к SQL-серверам и выполнять SQL-запросы так же, как это делает клиент MySQL или PostgreSQL. Операторы SQL доступны как Lua-методы. Таким образом, Tarantool может служить Lua-коннектором для MySQL или Lua-коннектором для PostgreSQL, что было бы полезно, даже если бы Tarantool больше ничего не умел. Но конечно же, Tarantool также представляет собой СУБД, поэтому модуль используется для любых операций, таких как копирование и ускорение базы данных, которые максимально эффективно, если приложение может работать как с SQL, так и с Tarantool в пределах одной Lua-процедуры. Методы подключения / выборки / вставки / и т.д. аналогичны методам модуля *net.box*.

С точки зрения пользователя, модули для MySQL и PostgreSQL очень похожи, поэтому следующие разделы – «Пример для MySQL» и «Пример для PostgreSQL» – слегка избыточны.

#### Пример для MySQL

В данном примере предполагается, что установлены MySQL 5.5, MySQL 5.6 или MySQL 5.7. Последние версии MariaDB также подойдут, используется коннектор к MariaDB для C. Самым важным пакетом будет пакет для разработчиков клиента MySQL, который обычно называется libmysqlclient-dev. Наиболее важным файлом из этого пакета будет файл libmysqlclient.so или с похожим названием. Можно использовать `` find`` или `` whereis``, чтобы узнать, в каких директориях установлены эти файлы.

Также нужно будет установить библиотеку общего пользования Tarantool с драйвером для MySQL, загрузить ее и использовать для подключения к экземпляру MySQL-сервера. После этого можно передавать любой оператор MySQL на экземпляр сервера и получать результаты, включая наборы результатов.

### Установка

Проверьте инструкции по загрузке и установке бинарного пакета, которые применимы к среде, где установлен Tarantool. Помимо установки `tarantool`, установите `tarantool-dev`. Например, в Ubuntu добавьте строку:

```
$ sudo apt-get install tarantool-dev
```

Что касается библиотеки общего пользования с драйвером для MySQL, ее можно установить двумя способами:

### Из LuaRocks

Begin by installing luarocks and making sure that tarantool is among the upstream servers, as in the instructions on rocks.tarantool.org, the Tarantool luarocks page. Now execute this:

```
luarocks install mysql [MYSQL_LIBDIR = path]
                       [MYSQL_INCDIR = path]
                       [--local]
```

Пример:

```
$ luarocks install mysql MYSQL_LIBDIR=/usr/local/mysql/lib
```

### Из GitHub

Go the site github.com/tarantool/mysql. Follow the instructions there, saying:

```
$ git clone https://github.com/tarantool/mysql.git
 $ cd mysql && cmake . -DCMAKE_BUILD_TYPE=RelWithDebInfo
 $ make
 $ make install
```

На данном этапе желательно проверить, что после установки появился файл под названием `driver.so`, а также проверить, что этот файл находится в директории, которую можно найти по запросу `require`.

### Подключение

Начните с выполнения запроса `require` для драйвера mysql. В дальнейших примерах у него будет имя `mysql`.

```
mysql = require('mysql')
```

Теперь выполните:

```
*имя_подключения* = mysql.connect(*параметры подключения*)
```

Параметры подключения включены в таблицу. Доступные параметры:

- `host = `*`имя-хоста`* – строка, значение по умолчанию = „localhost“
- `port = `*`номер-порта`* – число, значение по умолчанию = 3306

- `user = имя-пользователя` – строка, значение по умолчанию – имя пользователя в операционной системе
- `password = пароль` – строка, по умолчанию пустая
- `db = имя-базы-данных` – строка, по умолчанию пустая
- `raise = true/false` – логическое значение, по умолчанию, false (ложь)

The option names, except for `raise`, are similar to the names that MySQL's mysql client uses, for details see the MySQL manual at dev.mysql.com/doc/refman/5.6/en/connecting.html. The `raise` option should be set to true if errors should be raised when encountered. To connect with a Unix socket rather than with TCP, specify `host = 'unix/'` and `port = socket-name`.

Пример с использованием таблицы, заключенной в {фигурные скобки}:

```
conn = mysql.connect({
     host = '127.0.0.1',
     port = 3306,
     user = 'p',
     password = 'p',
     db = 'test',
     raise = true
 })
-- ИЛИ
conn = mysql.connect({
    host = 'unix/',
    port = '/var/run/mysqld/mysqld.sock'
})
```

Пример с созданием функции, которая определяет параметры в отдельных строках:

```
tarantool> -- Функция подключения. Использование: conn = mysql_connect()
tarantool> function mysql_connection()
         >   local p = {}
         >   p.host = 'widgets.com'
         >   p.db = 'test'
         >   conn = mysql.connect(p)
         >   return conn
         > end
---
...
tarantool> conn = mysql_connect()
---
...
```

Предполагаем, что в дальнейших примерах будет использоваться имя „conn“.

### Как проверить связь

Чтобы убедиться, что подключение работает, следует использовать запрос:

```
*имя-соединение*:ping()
```

**Пример:**

```
tarantool> conn:ping()
---
- true
...
```

### Исполнение оператора

Для всех операторов MySQL запрос будет:

```
*имя-соединения*:execute(*sql-оператор* [, *параметры*])
```

где `sql-statement` – это строка, а необязательные параметры – это дополнительные значения, которыми можно заменить любые знаки вопроса («?») в SQL-операторе.

**Пример:**

```
tarantool> conn:execute('select table_name from information_schema.tables')
---
- - table_name: ALL_PLUGINS
  - table_name: APPLICABLE_ROLES
  - table_name: CHARACTER_SETS
  <...>
- 78
...
```

### Закрытие соединения

Чтобы закрыть сессию, которую открыли с помощью `mysql.connect`, используется следующий запрос:

```
*имя-соединения*:close()
```

**Пример:**

```
tarantool> conn:close()
---
...
```

For further information, including examples of rarely-used requests, see the README.md file at github.com/tarantool/mysql.

### Пример

Пример выполняется на машине с ОС Ubuntu 12.04 (Precise Pangolin), где Tarantool установлен в поддиректорию /usr, а копия MySQL установлена в ~/mysql-5.5. Экземпляр сервера mysqld уже запущен на localhost 127.0.0.1.

```
$ export TMDIR=~/mysql-5.5
$ # Check that the include subdirectory exists by looking
$ # for .../include/mysql.h. (If this fails, there's a chance
$ # that it's in .../include/mysql/mysql.h instead.)
$ [ -f $TMDIR/include/mysql.h ] && echo "OK" || echo "Error"
OK

$ # Check that the library subdirectory exists and has the
$ # necessary .so file.
$ [ -f $TMDIR/lib/libmysqlclient.so ] && echo "OK" || echo "Error"
OK

$ # Check that the mysql client can connect using some factory
$ # defaults: port = 3306, user = 'root', user password = '',
$ # database = 'test'. These can be changed, provided one uses
$ # the changed values in all places.
$ $TMDIR/bin/mysql --port=3306 -h 127.0.0.1 --user=root \
    --password= --database=test
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25
Server version: 5.5.35 MySQL Community Server (GPL)
...
Type 'help;' or '\h' for help. Type '\c' to clear ...

$ # Insert a row in database test, and quit.
mysql> CREATE TABLE IF NOT EXISTS test (s1 INT, s2 VARCHAR(50));
Query OK, 0 rows affected (0.13 sec)
mysql> INSERT INTO test.test VALUES (1,'MySQL row');
Query OK, 1 row affected (0.02 sec)
mysql> QUIT
Bye

$ # Install luarocks
$ sudo apt-get -y install luarocks | grep -E "Setting up|already"
Setting up luarocks (2.0.8-2) ...

$ # Set up the Tarantool rock list in ~/.luarocks,
$ # following instructions at rocks.tarantool.org
$ mkdir ~/.luarocks
$ echo "rocks_servers = {[[http://rocks.tarantool.org/]]}" >> \
    ~/.luarocks/config.lua

$ # Ensure that the next "install" will get files from Tarantool
$ # master repository. The resultant display is normal for Ubuntu
$ # 12.04 precise
$ cat /etc/apt/sources.list.d/tarantool.list
deb http://tarantool.org/dist/2.1/ubuntu/ precise main
```

(continues on next page)

(продолжение с предыдущей страницы)

```
deb-src http://tarantool.org/dist/2.1/ubuntu/ precise main

$ # Install tarantool-dev. The displayed line should show version = 2.1
$ sudo apt-get -y install tarantool-dev | grep -E "Setting up|already"
Setting up tarantool-dev (2.1.0.222.g48b98bb~precise-1) ...
$

$ # Use luarocks to install locally, that is, relative to $HOME
$ luarocks install mysql MYSQL_LIBDIR=/usr/local/mysql/lib --local
Installing http://rocks.tarantool.org/mysql-scm-1.rockspec...
... (more info about building the Tarantool/MySQL driver appears here)
mysql scm-1 is now built and installed in ~/.luarocks/

$ # Ensure driver.so now has been created in a place
$ # tarantool will look at
$ find ~/.luarocks -name "driver.so"
~/.luarocks/lib/lua/5.1/mysql/driver.so

$ # Change directory to a directory which can be used for
$ # temporary tests. For this example we assume that the name
$ # of this directory is /home/pgulutzan/tarantool_sandbox.
$ # (Change "/home/pgulutzan" to whatever is the user's actual
$ # home directory for the machine that's used for this test.)
$ cd /home/pgulutzan/tarantool_sandbox

$ # Start the Tarantool server instance. Do not use a Lua initialization file.

$ tarantool
tarantool: version 2.1.0-222-g48b98bb
type 'help' for interactive help
tarantool>
```

Настройте Tarantool и загрузите модуль mysql. Убедитесь, что Tarantool не выбрасывает ошибку в ответ на вызов «require()».

```
tarantool> box.cfg{}
...
tarantool> mysql = require('mysql')
---
...
```

Создайте Lua-функцию, которая подключится к экземпляру MySQL-сервера (используя значения по умолчанию для параметров порта, пользователя и пароля), выберите одну строку и выведите ее на экран. Описание используемых здесь типов операторов вы можете найти в практикуме по Lua в руководстве пользователя Tarantool.

```
tarantool> function mysql_select ()
         >   local conn = mysql.connect({
         >     host = '127.0.0.1',
         >     port = 3306,
         >     user = 'root',
         >     db = 'test'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
         >   })
         >   local test = conn:execute('SELECT * FROM test WHERE s1 = 1')
         >   local row = ''
         >   for i, card in pairs(test) do
         >       row = row .. card.s2 .. ' '
         >       end
         >   conn:close()
         >   return row
         > end
---
...
tarantool> mysql_select()
---
- 'MySQL row '
...
```

Просмотрите результат. В нем есть строка «MySQL row». Это и есть строка, которая была вставлена в базу данных MySQL. А сейчас она выделена с помощью Tarantool-клиента.

### Пример для PostgreSQL

В данном примере предполагается, что установлены PostgreSQL 8 или PostgreSQL 9. Более поздние версии также должны сработать. Самым важным пакетом будет пакет для разработчиков клиента PostgreSQL, который обычно называется libpq-dev. На Ubuntu его можно установить следующим образом:

```
$ sudo apt-get install libpq-dev
```

Однако, не все платформы одинаковы, поэтому в данном примере предполагается, что пользователь должен проверить наличие нужных PostgreSQL-файлов, а также явным образом прописать, где они находятся, для сборки драйвера Tarantool/PostgreSQL. Для поиска директорий, где установлены PostgreSQL-файлы, можно воспользоваться командами `find` или `whereis`.

Также нужно будет установить библиотеку общего пользования Tarantool с драйвером для PostgreSQL, загрузить ее и использовать для подключения к экземпляру PostgreSQL-сервера. После этого можно передавать любой оператор PostgreSQL на экземпляр сервера и получать результаты.

### Установка

Проверьте инструкции по загрузке и установке бинарного пакета, которые применимы к среде, где установлен Tarantool. Помимо установки `tarantool`, установите `tarantool-dev`. Например, в Ubuntu добавьте строку:

```
$ sudo apt-get install tarantool-dev
```

Что касается библиотеки общего пользования с драйвером для PostgreSQL, ее можно установить двумя способами:

### Из LuaRocks

Начните с установки luarocks. Убедитесь, что tarantool указан в серверах, как описано на странице сторонних модулей Tarantool rocks.tarantool.org. Затем выполните:

```
luarocks install pg [POSTGRESQL_LIBDIR = *путь*]
                    [POSTGRESQL_INCDIR = *путь*]
                    [--local]
```

Пример:

```
$ luarocks install pg POSTGRESQL_LIBDIR=/usr/local/postgresql/lib
```

### Из GitHub

Перейдите по ссылке github.com/tarantool/pg. Следуя инструкциям, введите команду:

```
$ git clone https://github.com/tarantool/pg.git
$ cd pg && cmake . -DCMAKE_BUILD_TYPE=RelWithDebInfo
$ make
$ make install
```

На данном этапе желательно проверить, что после установки появился файл под названием `driver.so`, а также проверить, что этот файл находится в директории, которую можно найти по запросу `require`.

### Подключение

Начните с выполнения запроса `require` для драйвера pg. В дальнейших примерах у него будет имя `pg`.

```
pg = require('pg')
```

Теперь выполните:

```
*имя_подключения* = pg.connect(*параметры подключения*)
```

Параметры подключения включены в таблицу. Доступные параметры:

- `host = `*`имя-хоста`* – строка, значение по умолчанию = „localhost“
- `port = `*`номер-порта`* – число, значение по умолчанию = 5432
- `user = `*`имя-пользователя`* – строка, значение по умолчанию – имя пользователя в операционной системе
- `pass = `*`пароль`* или `password = `*`пароль`* – строка, по умолчанию пустая
- `db = `*`имя-базы-данных`* – строка, по умолчанию пустая

Имена параметров похожи на имена, которые используются в PostgreSQL.

Пример с использованием таблицы, заключенной в {фигурные скобки}:

```
conn = pg.connect({
    host = '127.0.0.1',
    port = 5432,
    user = 'p',
    password = 'p',
    db = 'test'
})
```

Пример с созданием функции, которая определяет параметры в отдельных строках:

```
tarantool> function pg_connect()
         >   local p = {}
         >   p.host = 'widgets.com'
         >   p.db = 'test'
         >   p.user = 'postgres'
         >   p.password = 'postgres'
         >   local conn = pg.connect(p)
         >   return conn
         > end
---
...
tarantool> conn = pg_connect()
---
...
```

Предполагаем, что в дальнейших примерах будет использоваться имя „conn“.

#### Как проверить связь

Чтобы убедиться, что подключение работает, следует использовать запрос:

```
*имя-соединение*:ping()
```

**Пример:**

```
tarantool> conn:ping()
---
- true
...
```

#### Исполнение оператора

Для всех операторов PostgreSQL запрос будет:

```
*имя-соединения*:execute(*sql-оператор* [, *параметры*])
```

где `sql-statement` – это строка, а необязательные параметры – это дополнительные значения, которыми можно заменить любые местозаполнители ($1 $2 $3 и т.д.) в SQL-операторе.

**Пример:**

```
tarantool> conn:execute('select tablename from pg_tables')
---
- - tablename: pg_statistic
  - tablename: pg_type
  - tablename: pg_authid
  <...>
...
```

### Закрытие соединения

Чтобы закрыть сессию, которую открыли с помощью pg.connect, используется следующий запрос:

```
*имя-соединения*:close()
```

**Пример:**

```
tarantool> conn:close()
---
...
```

For further information, including examples of rarely-used requests, see the README.md file at github.com/tarantool/pg.

### Пример

Пример выполняется на машине с ОС Ubuntu 12.04 (Precise Pangolin), где Tarantool установлен в поддиректорию /usr, а копия PostgreSQL установлена в /usr. Экземпляр сервера PostgreSQL уже запущен на localhost 127.0.0.1.

```
$ # Check that the include subdirectory exists
$ # by looking for /usr/include/postgresql/libpq-fe-h.
$ [ -f /usr/include/postgresql/libpq-fe.h ] && echo "OK" || echo "Error"
OK

$ # Check that the library subdirectory exists and has the necessary .so file.
$ [ -f /usr/lib/x86_64-linux-gnu/libpq.so ] && echo "OK" || echo "Error"
OK

$ # Check that the psql client can connect using some factory defaults:
$ # port = 5432, user = 'postgres', user password = 'postgres',
$ # database = 'postgres'. These can be changed, provided one changes
$ # them in all places. Insert a row in database postgres, and quit.
$ psql -h 127.0.0.1 -p 5432 -U postgres -d postgres
Password for user postgres:
psql (9.3.10)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=# CREATE TABLE test (s1 INT, s2 VARCHAR(50));
CREATE TABLE
postgres=# INSERT INTO test VALUES (1,'PostgreSQL row');
```

(continues on next page)

(продолжение с предыдущей страницы)

```
INSERT 0 1
postgres=# \q
$

$ # Install luarocks
$ sudo apt-get -y install luarocks | grep -E "Setting up|already"
Setting up luarocks (2.0.8-2) ...

$ # Set up the Tarantool rock list in ~/.luarocks,
$ # following instructions at rocks.tarantool.org
$ mkdir ~/.luarocks
$ echo "rocks_servers = {[[http://rocks.tarantool.org/]]}" >> \
        ~/.luarocks/config.lua

$ # Ensure that the next "install" will get files from Tarantool master
$ # repository. The resultant display is normal for Ubuntu 12.04 precise
$ cat /etc/apt/sources.list.d/tarantool.list
deb http://tarantool.org/dist/2.0/ubuntu/ precise main
deb-src http://tarantool.org/dist/2.0/ubuntu/ precise main

$ # Install tarantool-dev. The displayed line should show version = 2.0
$ sudo apt-get -y install tarantool-dev | grep -E "Setting up|already"
Setting up tarantool-dev (2.0.4.222.g48b98bb~precise-1) ...
$

$ # Use luarocks to install locally, that is, relative to $HOME
$ luarocks install pg POSTGRESQL_LIBDIR=/usr/lib/x86_64-linux-gnu --local
Installing http://rocks.tarantool.org/pg-scm-1.rockspec...
... (more info about building the Tarantool/PostgreSQL driver appears here)
pg scm-1 is now built and installed in ~/.luarocks/

$ # Ensure driver.so now has been created in a place
$ # tarantool will look at
$ find ~/.luarocks -name "driver.so"
~/.luarocks/lib/lua/5.1/pg/driver.so

$ # Change directory to a directory which can be used for
$ # temporary tests. For this example we assume that the
$ # name of this directory is $HOME/tarantool_sandbox.
$ # (Change "$HOME" to whatever is the user's actual
$ # home directory for the machine that's used for this test.)
cd $HOME/tarantool_sandbox

$ # Start the Tarantool server instance. Do not use a Lua initialization file.

$ tarantool
tarantool: version 2.0.4-412-g803b15c
type 'help' for interactive help
tarantool>
```

Настройте Tarantool и загрузите модуль pg. Убедитесь, что Tarantool не выбрасывает ошибку в ответ на вызов «require()».

```
tarantool> box.cfg{}
...
tarantool> pg = require('pg')
---
...
```

Создайте Lua-функцию, которая подключится к PostgreSQL-серверу (используя значения по умолчанию для параметров порта, пользователя и пароля), выберите одну строку и выведите ее на экран. Описание используемых здесь типов операторов вы можете найти в практикуме по Lua в руководстве пользователя Tarantool.

```
tarantool> function pg_select ()
         >   local conn = pg.connect({
         >     host = '127.0.0.1',
         >     port = 5432,
         >     user = 'postgres',
         >     password = 'postgres',
         >     db = 'postgres'
         >   })
         >   local test = conn:execute('SELECT * FROM test WHERE s1 = 1')
         >   local row = ''
         >   for i, card in pairs(test) do
         >       row = row .. card.s2 .. ' '
         >       end
         >   conn:close()
         >   return row
         > end
---
...
tarantool> pg_select()
---
- 'PostgreSQL row '
...
```

Просмотрите результат. В нем есть строка «PostgreSQL row». Это и есть строка, которая была вставлена в базу данных PostgreSQL. А сейчас она выделена с помощью Tarantool-клиента.

### 9.6.5 Other rocks

This page features a list of links to third-party Tarantool module documentation that is hosted externally – mostly on GitHub pages or in READMEs:

- crud
- expiration
- icu-date
- tracing

For Tarantool Enterprise modules, see the *Tarantool EE* documentation.

For Tarantool Enterprise modules, see the *Tarantool EE* documentation.

## 9.7 Справочник по C API

### 9.7.1 Module box

type box_function_ctx_t
: Непрозрачная структура, передаваемая в хранимую процедуру на языке C.

int box_return_tuple(*box_function_ctx_t* *ctx, *box_tuple_t* *tuple)
: Возврат кортежа с помощью хранимой процедуры на языке C.

  Для возвращаемого кортежа Tarantool проводит автоматический подсчет ссылок. Пример программы, которая использует box_return_tuple(): *write.c*.

  **Параметры**

  - ctx (*box_function_ctx_t* *) – непрозрачная структура, передаваемая Tarantool в хранимую процедуру на языке C
  - tuple (*box_tuple_t* *) – возвращаемый кортеж

  **Результат** -1 в случае ошибки (возможная нехватка памяти; проверьте *box_error_last()*)

  **Результат** 0 в остальных случаях

int box_return_mp(*box_function_ctx_t* *ctx, const char *mp, const char *mp_end)
: Возврат указателя на серию байтов в формате MessagePack.

  Можно использовать вместо *box_return_tuple()*: передаст то же значение, но в формате MessagePack, а не в виде кортежа. Это может быть проще, чем box_return_tuple(), если результат будет небольшим, например, число, логическое значение или короткая строка. Кроме того, это экономит время в отличие от box_return_tuple(), если в результате пользователю не приходится создавать кортеж каждый раз, когда он хочет вернуть что-то из функции на C.

  С другой стороны, если из итератора получен уже существующий кортеж, то быстрее будет вернуть кортеж через box_return_tuple(), чем извлекать его части и отправлять их через box_return_mp().

  **Параметры**

  - ctx (*box_function_ctx_t* *) – непрозрачная структура, передаваемая Tarantool в хранимую процедуру на языке C
  - mp (char*) – первый байт в MessagePack
  - mp_end (char*) – после последнего байта в MessagePack

  **Результат** -1 в случае ошибки (возможная нехватка памяти; проверьте *box_error_last()*)

  **Результат** 0 в остальных случаях

  Например, если mp — это буфер, а mp_end — возвращаемое значение, полученное путем кодирования одного скалярного значения MP_UINT с помощью mp_end=mp_encode_uint(mp,1);, то box_return_mp(ctx,mp,mp_end); вернет 0.

uint32_t box_space_id_by_name(const char *name, uint32_t len)
: Поиск идентификатора спейса по имени.

  Данная функция делает запрос выборки SELECT из системного спейса _vspace.

  **Параметры**

- `name` (`const char*`) – имя спейса
- `len` (`uint32_t`) – длина имени `name`

**Результат** `BOX_ID_NIL` в случае ошибки или отсутствия (проверьте *box_error_last()*)

**Результат** space_id в остальных случаях

См. также *box_index_id_by_name*

uint32_t `box_index_id_by_name`(uint32_t space_id, const char *name, uint32_t len)
Поиск идентификатора индекса по имени.

Данная функция делает запрос выборки SELECT из системного спейса `_vindex`.

**Параметры**

- `space_id` (`uint32_t`) – идентификатор спейса
- `name` (`const char*`) – имя индекса
- `len` (`uint32_t`) – длина имени `name`

**Результат** `BOX_ID_NIL` в случае ошибки или отсутствия (проверьте *box_error_last()*)

**Результат** space_id в остальных случаях

См. также *box_space_id_by_name*

int `box_insert`(uint32_t space_id, const char *tuple, const char *tuple_end, *box_tuple_t* **result)
Выполнение запроса вставки или замены (INSERT/REPLACE).

**Параметры**

- `space_id` (`uint32_t`) – идентификатор спейса
- `tuple` (`const char*`) – закодированный кортеж в формате MsgPack-массива ([ field1, field2, ... ])
- `tuple_end` (`const char*`) – конец кортежа `tuple`
- `result` (*box_tuple_t*`**`) – аргумент вывода. Возвращаемый кортеж. Можно задать значение NULL для сброса результата

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0 в остальных случаях

See also: *space_object.insert()*

int `box_replace`(uint32_t space_id, const char *tuple, const char *tuple_end, *box_tuple_t* **result)
Выполнение запроса замены (REPLACE).

**Параметры**

- `space_id` (`uint32_t`) – идентификатор спейса
- `tuple` (`const char*`) – закодированный кортеж в формате MsgPack-массива ([ field1, field2, ... ])
- `tuple_end` (`const char*`) – конец кортежа `tuple`
- `result` (*box_tuple_t*`**`) – аргумент вывода. Возвращаемый кортеж. Можно задать значение NULL для сброса результата

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0 в остальных случаях

See also: *space_object.replace()*

int box_delete(uint32_t space_id, uint32_t index_id, const char *key, const char *key_end, *box_tuple_t* **result)

Выполнение запроса удаления (DELETE).

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса
- key (const char*) – закодированный ключ в формате MsgPack-массива ([ field1, field2, ...])
- key_end (const char*) – конец ключа key
- result (*box_tuple_t***) – аргумент вывода. Старый кортеж. Можно задать значение NULL для сброса результата

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0 в остальных случаях

See also: *space_object.delete()*

int box_update(uint32_t space_id, uint32_t index_id, const char *key, const char *key_end, const char *ops, const char *ops_end, int index_base, *box_tuple_t* **result)

Выполнение запроса обновления (UPDATE).

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса
- key (const char*) – закодированный ключ в формате MsgPack-массива ([ field1, field2, ...])
- key_end (const char*) – конец ключа key
- ops (const char*) – закодированные операции в формате MsgPack-массива, например [[ '=', field_id, value ], ['!', 2, 'xxx']]
- ops_end (const char*) – конец раздела операций ops
- index_base (int) – 0, если идентификаторы полей field_id с основанием 0, как в C, 1, если идентификаторы полей с основанием 1, как в Lua
- result (*box_tuple_t***) – аргумент вывода. Старый кортеж. Можно задать значение NULL для сброса результата

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0 в остальных случаях

See also: *space_object.update()*

int box_upsert(uint32_t space_id, uint32_t index_id, const char *tuple, const char *tuple_end, const char *ops, const char *ops_end, int index_base, *box_tuple_t* **result)

Выполнение запроса обновления и вставки (UPSERT).

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса

- `tuple` (`const char*`) – закодированный кортеж в формате MsgPack-массива ([ field1, field2, ... ])
- `tuple_end` (`const char*`) – конец кортежа `tuple`
- `ops` (`const char*`) – закодированные операции в формате MsgPack-массива, например `[[ '=', field_id, value ], ['!', 2, 'xxx']]`
- `ops_end` (`const char*`) – конец операций `ops`
- `index_base` (`int`) – 0, если идентификаторы полей field_id с основанием 0, как в C, 1, если идентификаторы полей с основанием 1, как в Lua
- `result` (*box_tuple_t*`**`) – аргумент вывода. Старый кортеж. Можно задать значение NULL для сброса результата

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0 в остальных случаях

See also: *space_object.upsert()*

int `box_truncate`(uint32_t space_id)
Очистка спейса.

**Параметры**

- `space_id` (`uint32_t`) – идентификатор спейса

int `box_session_push`(const char *data, const char *data_end)
С версии *2.4.1*. Передача данных в формате MessagePack в канал данных сеанса — сокет, консоль или то, что используется в сеансе. Работает, как *box.session.push()* в Lua.

**Параметры**

- `data` (`const char*`) – начало данных для передачи
- `data_end` (`const char*`) – конец данных для передачи

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0 в остальных случаях

int `box_sequence_current`(uint32_t seq_id, int64_t *result)
С версии *2.4.1*. Возврат последнего полученного значения из указанной последовательности.

**Параметры**

- `seq_id` (`uint32_t`) – идентификатор последовательности
- `result` (`int64_t`) – указатель на переменную, где будет храниться значение последовательности в случае успеха.

**Результат** 0 при успехе и -1 в обратном случае. В случае ошибки, ее можно получить с помощью `box_error_last()`.

uint32_t `box_schema_version`(void)
Since version *2.11.0*. Return the database schema version. A schema version is a number that indicates whether the *database schema* is changed. For example, the `schema_version` value grows if a *space* or *index* is added or deleted, or a space, index, or field name is changed.

**Результат** the database schema version

**Тип результата** number

See also: *box.info.schema_version* and *IPROTO_SCHEMA_VERSION*

uint64_t box_session_id(void)
Since version *2.11.0*. Return the unique identifier (ID) for the current session.

**Результат** the session identifier; 0 or -1 if there is no session

**Тип результата** number

See also: *box.session.id()*

int box_iproto_send(uint64_t sid, char *header, char *header_end[, char *body, char *body_end])
Since version *2.11.0*. Send an *IPROTO* packet over the session's socket with the given MsgPack header and body. The function yields. The function works for binary sessions only. For details, see *box.session.type()*.

**Параметры**

- `sid` (`uint32_t`) – the IPROTO session identifier (see *box_session_id()*)
- `header` (`char*`) – a MsgPack-encoded header
- `header_end` (`char*`) – end of a header encoded as MsgPack
- `body` (`char*`) – a MsgPack-encoded body. If the `body` and `body_end` parameters are omitted, the packet consists of the header only.
- `body_end` (`char*`) – end of a body encoded as MsgPack

**Результат** 0 on success; -1 on error (check *box_error_last()*)

**Тип результата** number

See also: *box.iproto.send()*

**Possible errors:**

- `ER_SESSION_CLOSED` – the session is closed.
- `ER_NO_SUCH_SESSION` – the session does not exist.
- `ER_MEMORY_ISSUE` – out-of-memory limit has been reached.
- `ER_WRONG_SESSION_TYPE` – the session type is not binary.

For details, see src/box/errcode.h.

**Example**

```
/* IPROTO constants are not exported to C.
* That is, the user encodes them by himself.
*/
#define IPROTO_REQUEST_TYPE 0x00
#define IPROTO_OK 0x00
#define IPROTO_SYNC 0x01
#define IPROTO_SCHEMA_VERSION 0x05
#define IPROTO_DATA 0x30

char buf[256] = {};
char *header = buf;
char *header_end = header;
header_end = mp_encode_map(header_end, 3);
header_end = mp_encode_uint(header_end, IPROTO_REQUEST_TYPE);
header_end = mp_encode_uint(header_end, IPROTO_OK);
header_end = mp_encode_uint(header_end, IPROTO_SYNC);
```

(continues on next page)

(продолжение с предыдущей страницы)

```
header_end = mp_encode_uint(header_end, 10);
header_end = mp_encode_uint(header_end, IPROTO_SCHEMA_VERSION);
header_end = mp_encode_uint(header_end, box_schema_version());

char *body = header_end;
char *body_end = body;
body_end = mp_encode_map(body_end, 1);
body_end = mp_encode_uint(body_end, IPROTO_DATA);
body_end = mp_encode_uint(body_end, 1);

/* The packet contains both the header and body. */
box_iproto_send(box_session_id(), header, header_end, body, body_end);

/* The packet contains the header only. */
box_iproto_send(box_session_id(), header, header_end, NULL, NULL);
```

void box_iproto_override(uint32_t request_type, iproto_handler_t handler,
iproto_handler_destroy_t destroy, void *ctx)

Since version *2.11.0*. Set a new IPROTO request handler with the provided context for the given request type. The function yields.

**Параметры**

- `request_type` (`uint32_t`) – IPROTO request type code (for example, `IPROTO_SELECT`). For details, check *Client-server requests and responses*.

  To override the handler of unknown request types, use the IPROTO_UNKNOWN type code.

- `handler` (`iproto_handler_t`) – IPROTO request handler. To reset the request handler, set the `handler` parameter to `NULL`. See the full parameter description in the *Handler function* section.

- `destroy` (`iproto_handler_destroy_t`) – IPROTO request handler destructor. The destructor is called when the corresponding handler is removed. See the full parameter description in the *Handler destructor function* section.

- `ctx` (`void*`) – a context passed to the `handler` and `destroy` callbacks

**Результат** 0 on success; -1 on error (check *box_error_last()*)

**Тип результата** number

See also: *box.iproto.override()*

**Possible errors:**

If a Lua handler throws an exception, the behavior is similar to that of a remote procedure call. The following errors are returned to the client over IPROTO (see src/lua/utils.h):

- `ER_PROC_LUA` – an exception is thrown from a Lua handler, diagnostic is not set.
- diagnostics from `src/box/errcode.h` – an exception is thrown, diagnostic is set.

For details, see src/box/errcode.h.

**Handler function**

The signature of a handler function (the `handler` parameter):

```
enum iproto_handler_status {
        IPROTO_HANDLER_OK,
        IPROTO_HANDLER_ERROR,
        IPROTO_HANDLER_FALLBACK,
}

typedef enum iproto_handler_status
(*iproto_handler_t)(const char *header, const char *header_end,
                    const char *body, const char *body_end, void *ctx);
```

where:

- `header` (const char*) – a MsgPack-encoded header
- `header_end` (const char*) – end of a header encoded as MsgPack
- `body` (const char*) – a MsgPack-encoded body
- `header_end` (const char*) – end of a body encoded as MsgPack

The handler returns a status code. Possible statuses:

- `IPROTO_REQUEST_HANDLER_OK` – success
- `IPROTO_REQUEST_HANDLER_ERROR` – error, diagnostic must be set by handler (see *box_error_set()* and *box_error_raise()*)
- `IPROTO_REQUEST_HANDLER_FALLBACK` – fallback to the default handler

**Handler destructor function**

The destructor is called when the handler is reset. The signature of a destructor function (the `destroy` parameter):

```
typedef void (*iproto_handler_destroy_t)(void *ctx);
```

where:

- `ctx` (void*): the context provided by `box_iproto_override()` function.

**Examples**

```
box_iproto_override(1000, iproto_request_handler_c, NULL)
box_iproto_override(IPROTO_SELECT, iproto_request_handler_c, (uintptr_t)23)
box_iproto_override(IPROTO_SELECT, NULL, NULL)
box_iproto_override(IPROTO_UNKNOWN, iproto_unknown_request_handler_c, &ctx)
```

### 9.7.2 Module clock

double `clock_realtime`(void)

double `clock_monotonic`(void)

double `clock_process`(void)

double `clock_thread`(void)

int64_t `clock_realtime64`(void)

int64_t `clock_monotonic64`(void)

int64_t `clock_process64`(void)

int64_t `clock_thread64`(void)

### 9.7.3 Module coio

enum `COIO_EVENT`

enumerator ::`COIO_READ`
: событие чтения READ

enumerator ::`COIO_WRITE`
: событие записи WRITE

int `coio_wait`(int fd, int event, double timeout)
: Ожидание события чтения или записи (READ / WRITE) на сокете (`fd`) с передачей управления.

**Параметры**

- `fd` (`int`) – дескриптор файла сокета без блокировки
- `event` (`int`) – запрашиваемые события. Комбинация битовых флагов `COIO_READ | COIO_WRITE`.
- `timeout` (`double`) – время ожидания в секундах.

**Результат** 0 - время ожидания

**Результат** >0 - возвращаемые события. Комбинация битовых флагов `TNT_IO_READ | TNT_IO_WRITE`.

ssize_t `coio_call`(ssize_t (*func)(va_list), ...)
: Создание новой задачи для eio с указанной функцией и аргументами. Передает управление и ожидает окончания задачи. Функция может использовать конфигурационный параметр *worker_pool_threads*.

Во избежание двойной проверки ошибок функция не выбрасывает исключения. В большинстве случаев также необходимо проверять возвращаемое значение вызванной функции и выполнить необходимые действия. Если функция определяет номер ошибки errno, этот номер ошибки сохраняется в течение вызова.

**Результат** -1 и `errno` = ENOMEM, если задача не была создана

**Результат** возврат функции (`errno` сохраняется).

**Пример:**

```
static ssize_t openfile_cb(va_list ap)
{
        const char* filename = va_arg(ap);
        int flags = va_arg(ap);
        return open(filename, flags);
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
if (coio_call(openfile_cb, "/tmp/file", 0) == -1)
    // обработка ошибок.
...
```

int coio_getaddrinfo(const char *host, const char *port, const struct addrinfo *hints, struct addrinfo **res, double timeout)

Вариант функции getaddrinfo(3), совместимый с файберами.

int coio_close(int fd)

Закрытие `fd` и пробуждение любого файбера, заблокированного в вызове *coio_wait()* на данном сокете `fd`.

**Параметры**

- `fd` (`int`) – дескриптор файла сокета без блокировки

**Результат** результат `close(fd)`, см. *close(2)*

### 9.7.4 Module error

enum box_error_code

For a complete list of errors, refer to the Tarantool error code header file.

type box_error_t

Ошибка – содержит информацию об ошибке.

const char *box_error_type(const *box_error_t* *error)

Возврат типа ошибки, например, «ClientError», «SocketError» и т.д.

**Параметры**

- `error` (*box_error_t* *) – ошибка

**Результат** ненулевая строка

uint32_t box_error_code(const *box_error_t* *error)

Возврат кода ошибки IPROTO

**Параметры**

- `error` (*box_error_t* *) – ошибка

**Результат** *box_error_code*

const char *box_error_message(const *box_error_t* *error)

Возврат сообщения ошибки

**Параметры**

- `error` (*box_error_t* *) – ошибка

**Результат** ненулевая строка

*box_error_t* *box_error_last(void)

Получение информации о последней ошибке вызова API.

Обработка ошибок в Tarantool больше всего похожа на errno в стандартной библиотеке языка C libc. Все вызовы API возвращают -1 или NULL в случае ошибки. Внутренний указатель на тип box_error_t задается функциями, чтобы указать, что пошло не так. Это значение показательно, если вызов API не прошел (вернулось -1 или NULL).

Выполненная функция в некоторых случаях также может затрагивать последнюю ошибку. Необязательно удалять последнюю ошибку перед вызовом API-функций. Возвращаемый объект применим только до следующего вызова **любой** API-функции.

Следует задать последнюю ошибку с помощью box_error_set() из хранимых процедур на языке C, если необходимо вернуть специальное сообщение об ошибке. Можно повторно сгенерировать последнюю API-ошибку в клиент IPROTO, сохранив текущее значение и вернув -1 to Tarantool из хранимой процедуры.

**Результат** последняя ошибка

void `box_error_clear`(void)
Удаление последней ошибки.

int `box_error_set`(const char *file, unsigned line, uint32_t code, const char *format, ...)
Определение последней ошибки.

**Параметры**

- `file` (`const char*`) –
- `line` (`unsigned`) –
- `code` (`uint32_t`) – IPROTO *error code*
- `format` (`const char*`) –
- `...` – аргументы формата

См. также IPROTO *error code*

`box_error_raise`(code, format, ...)
Обратно совместимые определения API.

### 9.7.5 Module fiber

type `fiber`
Fiber - contains information about a *fiber*.

typedef int (*`fiber_func`)(va_list)
Функции для выполнения в файбере.

*fiber* *`fiber_new`(const char *name, *fiber_func* f)
Создание нового файбера.

Берет файбер из кэша файберов, если в нем что-то есть. Может не сработать, только если недостаточно памяти для структуры файбера или стека файбера.

Созданный файбер автоматически возвращается в кэш файберов, когда выполнена его основная функция.

**Параметры**

- `name` (`const char*`) – строка с именем файбера
- `f` (*`fiber_func`*) – функция для выполнения в файбере

См. также *fiber_start()*

*fiber* *`fiber_new_ex`(const char *name, const struct *fiber_attr* *fiber_attr, *fiber_func* f)
Создание нового файбера с заданными атрибутами.

Может не сработать, только если недостаточно памяти для структуры файбера или стека файбера.

Созданный файбер автоматически возвращается в кэш файберов, если у него размер стека по умолчанию, когда выполнена его основная функция.

**Параметры**

- `name` (`const char*`) – строка с именем файбера
- `fiber_attr` (`const struct` *fiber_attr*`*`) – контейнер с атрибутами файбера
- `f` (*fiber_func*) – функция для выполнения в файбере

См. также *fiber_ start()*

void `fiber_start`(struct *fiber* *callee, ...)
Запуск созданного файбера.

**Параметры**

- `callee` (`struct` *fiber*`*`) – запускаемый файбер
- `...` – аргументы для запуска файбера

void `fiber_yield`(void)
Передача управления другому файберу и ожидание его пробуждения.

См. также *fiber_ wakeup()*

void `fiber_wakeup`(struct *fiber* *f)
Прерывание синхронного ожидания файбера

**Параметры**

- `f` (`struct` *fiber*`*`) – пробуждаемый файбер

void `fiber_cancel`(struct *fiber* *f)
Отмена файбера.

Отмена файбера происходит асинхронно. Чтобы дождаться окончания отмены, используйте *fiber_join()*.

После вызова `fiber_cancel()` файбер может проверить, был ли он отменен. Если он этого не сделает, его отменить невозможно.

**Параметры**

- `f` (`struct` *fiber*`*`) – отменяемый файбер

bool `fiber_set_cancellable`(bool yesno)
Deprecated since *2.11.0*. Make it possible or not possible to wakeup the current fiber immediately when it's cancelled.

**Параметры**

- `f` (`struct` *fiber*`*`) – файбер
- `yesno` (`bool`) – назначаемый статус

**Результат** предыдущий статус

void `fiber_set_joinable`(struct *fiber* *fiber, bool yesno)
Определение файбера как присоединяемого (по умолчанию `false`)

**Параметры**

- `f` (`struct` *fiber*`*`) – файбер
- `yesno` (`bool`) – назначаемый статус

void `fiber_join`(struct *fiber* *f)
Ожидание удаления файбера, а затем передача статуса его выполнения вызывающему клиенту. Файбер не должен быть откреплённым.

**Параметры**

- `f` (`struct` *`fiber`**) – пробуждаемый файбер

Ранее: установлен флаг `FIBER_IS_JOINABLE`.

См. также *fiber_set_joinable()*

void `fiber_sleep`(double s)
Перевод текущего файбера в режим ожидания как минимум на „s“ секунд.

**Параметры**

- `s` (`double`) – время ожидания

Примечание: это и есть точка отмены.

См. также *fiber_is_cancelled()*

bool `fiber_is_cancelled`(void)
Проверка отмены текущего файбера (это делается вручную).

double `fiber_time`(void)
Сообщение времени начала цикла в виде числа двойной точности.

int64_t `fiber_time64`(void)
Report loop begin time as 64-bit int. Uses real time clock.

double `fiber_clock`(void)
Report loop begin time as double (cheap). Uses monotonic clock.

int64_t `fiber_clock64`(void)
Report loop begin time as 64-bit int. Uses monotonic clock.

void `fiber_reschedule`(void)
Перенос файбера для завершения событийного цикла.

type `slab_cache`

struct *slab_cache* *`cord_slab_cache`(void)
Возврат `slab_cache`, подходящего для использования с библиотекой `tarantool/small`

*fiber* *`fiber_self`(void)
Возврат текущего файбера.

type `fiber_attr`

void `fiber_attr_new`(void)
Создание нового контейнера с атрибутами файбера и его инициализация с параметрами по умолчанию.

Можно использовать для создания множества файберов: смена владельца не произойдет.

void `fiber_attr_delete`(struct *fiber_attr* *fiber_attr)
Удаление `fiber_attr` и освобождение всех выделенных ресурсов. Используется, когда есть файберы, созданные с данным атрибутом.

**Параметры**

- `fiber_attribute` (`struct` *`fiber_attr`**) – контейнер с атрибутами файбера

int fiber_attr_setstacksize(struct *fiber_attr* *fiber_attr, size_t stack_size)
Определение размера стека файбера в контейнере с атрибутами файбера.

**Параметры**

- fiber_attr (struct *fiber_attr**) – контейнер с атрибутами файбера
- stack_size (size_t) – размер стека для новых файберов (в байтах)

**Результат** 0, если выполнено

**Результат** -1, если не выполнено (если размер стека stack_size меньше минимально допустимого размера стека файбера)

size_t fiber_attr_getstacksize(struct *fiber_attr* *fiber_attr)
Получение размера стека файбера из контейнера с атрибутами файбера.

**Параметры**

- fiber_attr (struct *fiber_attr**) – контейнер с атрибутами файбера или NULL, по умолчанию

**Результат** размер стека (в байтах)

type fiber_cond
A conditional variable: a synchronization primitive that allow fibers in Tarantool's *cooperative multitasking* environment to yield until some predicate is satisfied.

Условия работы файбера поддерживают две основные операции – «wait» (ожидание) и «signal» (сигнал), – где «wait» откладывает выполнение файбера (то есть передает управление) до тех пор, пока не будет вызван «signal».

В отличие от pthread_cond, fiber_cond не требует функции-обертки в виде мьютекса или защелки.

struct *fiber_cond* *fiber_cond_new(void)
Создание новой условной переменной.

void fiber_cond_delete(*fiber_cond* *cond)
Удаление условной переменной.

Примечание: поведение не определено, если есть файберы, ожидающие условной переменной.

**Параметры**

- cond (struct *fiber_cond**) – удаляемая условная переменная

void fiber_cond_signal(struct *fiber_cond* *cond);
Пробуждение **одного** (любого) файбера, ожидающего условной переменной.

Не делает ничего, если нет ожидающих файберов.

**Параметры**

- cond (struct *fiber_cond**) – условная переменная

void fiber_cond_broadcast(struct *fiber_cond* *cond);
Пробуждение **всех** файберов, ожидающих условной переменной.

Не делает ничего, если нет ожидающих файберов.

**Параметры**

- cond (struct *fiber_cond**) – условная переменная

int `fiber_cond_wait_timeout`(struct *fiber_ cond* *cond, double timeout)
Приостановление выполнения текущего файбера (т.е. передача управления) до вызова *fiber_ cond_ signal()*.

Как и `pthread_cond`, `fiber_cond` может отправлять ложные сигналы пробуждения с помощью вызова *fiber_ wakeup()* или *fiber_ cancel()*. Настоятельно рекомендуется заключать вызовы данной функции в цикл и проверять предикат и *fiber_ is_ cancelled()* при каждой итерации.

**Параметры**

- `cond` (`struct` *fiber_ cond*`*`) – условная переменная
- `timeout` (`double`) – время ожидания в секундах

**Результат** 0 при вызове *fiber_ cond_ signal()* или ложном пробуждении

**Результат** -1 в случае ожидания, и задается код ошибки „TimedOut“ (истекло время ожидания)

int `fiber_cond_wait`(struct *fiber_ cond* *cond)
Ускоренный метод для *fiber_ cond_ wait_ timeout()*.

### 9.7.6 Module index

type `box_iterator_t`
Итератор спейса

enum `iterator_type`
Управление итерацией кортежей в индексе. Различные типы индексов поддерживают различные типы итераторов. Например, можно начать итерацию с определенного значения (ключ запроса), а затем получить все кортежи, ключи которых больше или равны (= GE) заданному ключу.

Если тип итератора не поддерживается выбранным типом индекса, конструктор итератора прекратит работу с ошибкой ER_UNSUPPORTED. Чтобы индекс можно было выбрать для первичного ключа, он должен поддерживать типы ITER_EQ и ITER_GE.

Значение ключа запроса NULL соответствует первому или последнему ключу в индексе, в зависимости от направления итерации (первый ключ для типов GE и GT, последний ключ для типов LE и LT). Таким образом, для итерации по всем кортежам в индексе можно использовать типы итерации ITER_GE или ITER_LE с начальным ключом, который равен NULL. Для ITER_EQ ключ не должен равняться NULL.

enumerator ::`ITER_EQ`
ключ == x в порядке возрастания

enumerator ::`ITER_REQ`
ключ == x в порядке убывания

enumerator ::`ITER_ALL`
все кортежи

enumerator ::`ITER_LT`
ключ < x

enumerator ::`ITER_LE`
ключ <= x

enumerator ::`ITER_GE`
ключ >= x

enumerator ::ITER_GT
: ключ > x

enumerator ::ITER_BITS_ALL_SET
: все биты из x заданы в ключе

enumerator ::ITER_BITS_ANY_SET
: задан хотя бы один бит из x

enumerator ::ITER_BITS_ALL_NOT_SET
: ни один бит не задан

enumerator ::ITER_OVERLAPS
: ключ пересекается с x

enumerator ::ITER_NEIGHBOR
: кортежи в порядке возрастания расстояния из указанной точки

*box_iterator_t* *box_index_iterator(uint32_t space_id, uint32_t index_id, int type, const char *key, const char *key_end)

Выделение и инициализация итератора для space_id, index_id.

Возвращаемый итератор следует удалить с помощью *box_iterator_free*.

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса
- type (int) – *iterator_type*
- key (const char*) – кодировка ключа в формате MsgPack-массива ([part1, part2, ...])
- key_end (const char*) – часть закодированного ключа key

**Результат** NULL в случае ошибки (проверьте *box_error_last()*)

**Результат** итератор в остальных случаях

См. также *box_iterator_next*, *box_iterator_free*

int box_iterator_next(*box_iterator_t* *iterator, *box_tuple_t* **result)

Получение следующего пункта из итератора iterator.

**Параметры**

- iterator (*box_iterator_t**) – итератор, возвращаемый *box_index_iterator*
- result (*box_tuple_t***) – аргумент вывода. Результатом будет кортеж или NULL, если данных больше нет.

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0 в случае выполнения. Отсутствие данных не является ошибкой.

void box_iterator_free(*box_iterator_t* *iterator)

Удаление и освобождение итератора.

**Параметры**

- iterator (*box_iterator_t**) – итератор, возвращаемый *box_index_iterator*

int iterator_direction(enum iterator_type type)

Определение направления заданного типа итератора: -1 для REQ, LT, LE, и +1 для всех остальных.

ssize_t box_index_len(uint32_t space_id, uint32_t index_id)
Возврат номера элемента в индексе.

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** >= 0 в остальных случаях

ssize_t box_index_bsize(uint32_t space_id, uint32_t index_id)
Возврат количества байтов памяти, используемых индексом.

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** >= 0 в остальных случаях

int box_index_random(uint32_t space_id, uint32_t index_id, uint32_t rnd, *box_tuple_t* **result)
Возврат случайного кортежа из индекса (используется для статистического анализа).

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса
- rnd (uint32_t) – случайное начальное число
- result (*box_tuple_t***) – аргумент вывода. Результатом будет кортеж или NULL, если в спейсе нет кортежей.

См. также *index_object:random()*

int box_index_get(uint32_t space_id, uint32_t index_id, const char *key, const char *key_end,
*box_tuple_t* **result)
Получение кортежа из индекса по ключу.

Следует отметить, что данная функция работает намного быстрее, чем *index_object:select()* или *box_index_iterator* + *box_iterator_next*.

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса
- key (const char*) – кодировка ключа в формате MsgPack-массива ([part1, part2, ...])
- key_end (const char*) – часть закодированного ключа key
- result (*box_tuple_t***) – аргумент вывода. Результатом будет кортеж или NULL, если в спейсе нет кортежей.

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0, если выполнено

См. также index_object.get()

int box_index_min(uint32_t space_id, uint32_t index_id, const char *key, const char *key_end, *box_tuple_t* **result)

Возврат первого (минимального) кортежа, который соответствует заданному ключу.

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса
- key (const char*) – кодировка ключа в формате MsgPack-массива ([part1, part2, ...])
- key_end (const char*) – часть закодированного ключа key
- result (*box_tuple_t***) – аргумент вывода. Результатом будет кортеж или NULL, если в спейсе нет кортежей.

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0, если выполнено

См. также *index_object:min()*

int box_index_max(uint32_t space_id, uint32_t index_id, const char *key, const char *key_end, *box_tuple_t* **result)

Возврат последнего (максимального) кортежа, который соответствует заданному ключу.

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса
- key (const char*) – кодировка ключа в формате MsgPack-массива ([part1, part2, ...])
- key_end (const char*) – часть закодированного ключа key
- result (*box_tuple_t***) – аргумент вывода. Результатом будет кортеж или NULL, если в спейсе нет кортежей.

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0, если выполнено

См. также *index_object:max()*

ssize_t box_index_count(uint32_t space_id, uint32_t index_id, int type, const char *key, const char *key_end)

Подсчет количества кортежей, которые соответствуют заданному ключу.

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса
- type (int) – *iterator_type*
- key (const char*) – кодировка ключа в формате MsgPack-массива ([part1, part2, ...])
- key_end (const char*) – часть закодированного ключа key

**Результат** -1 в случае ошибки (проверьте *box_error_last()*)

**Результат** 0, если выполнено

См. также *index_object.count()*

const *box_key_def_t* *box_index_key_def(uint32_t space_id, uint32_t index_id)
Возврат *определения ключа* для индекса

Возвращаемый объект действителен до следующей передачи управления.

**Параметры**

- space_id (uint32_t) – идентификатор спейса
- index_id (uint32_t) – идентификатор индекса

**Результат** определение ключа, если выполнено

**Результат** NULL в случае ошибки

**См. также** ***box_tuple_compare()***, *box_tuple_format_new()*

### 9.7.7 Module latch

type box_latch_t
Блокировка среды кооперативной многозадачности

*box_latch_t* *box_latch_new(void)
Выделение и инициализация новой защелки.

**Результат** выделенная защелка

**Тип результата** box_latch_t *

void box_latch_delete(*box_latch_t* *latch)
Удаление и освобождение защелки.

**Параметры**

- latch (*box_latch_t**) – удаляемая защелка

void box_latch_lock(*box_latch_t* *latch)
Lock a latch. Waits indefinitely until the current fiber can gain access to the latch. Since version *2.11.0*, locks are acquired exactly in the order in which they were requested.

**Параметры**

- latch (*box_latch_t**) – применяемая защелка

int box_latch_trylock(*box_latch_t* *latch)
Попытка применить защелку. Возвращается незамедлительно, если защелка поставлена.

**Параметры**

- latch (*box_latch_t**) – применяемая защелка

**Результат** статус операции. 0 – успешно, 1 – защелка поставлена

**Тип результата** целое число

void box_latch_unlock(*box_latch_t* *latch)
Отмена защелки. Файбер, который вызывает данную функцию, должен иметь права на защелку.

**Параметры**

- latch (*box_latch_t**) – отменяемая защелка

### 9.7.8 Function on_shutdown

int box_on_shutdown(void *arg, int (*new_handler)(void*), int (*old_handler)(void*))
Register and/or deregister an on_shutdown function.

**Параметры**

- arg (void*) – Pointer to an area that the new handler can use
- new_handler (function*) – Pointer to a function which will be registered, or NULL
- old_handler (function*) – Pointer to a function which will be deregistered, or NULL

**Результат** status of operation. 0 - success, -1 - failure

**Тип результата** целое число

A function which is registered will be called when the Tarantool instance shuts down. This is functionally similar to what *box.ctl.on_shutdown* does.

If there are several on_shutdown functions, the Tarantool instance will call them in reverse order of registration, that is, it will call the last-registered function first.

Typically a module developer will register an on_shutdown function that does whatever cleanup work the module requires, and then returns control to the Tarantool instance. Such an on_shutdown function should be fast, or should use an asynchronous waiting mechanism (for example *coio_wait*).

Possible errors: old_handler does not exist (errno = EINVAL), new_handler and old_handler are both NULL (errno = EINVAL), memory allocation fails (errno = ENOMEM).

Example: if the C API .c program contains a function `int on_shutdown_function(void *arg) {printf("Bye!\n");return 0; }` and later, in the function which the instance calls, contains a line `box_on_shutdown(NULL, on_shutdown_function, NULL);` then, if all goes well, when the instance shuts down, it will display «Bye!».

Added in version *2.8.1*.

### 9.7.9 Module lua/utils

void *luaL_pushcdata(struct lua_State *L, uint32_t ctypeid)
Занесение cdata заданного ctypeid в стек.

CTypeID должен быть использован хотя бы один раз из FFI. Выделенная область памяти возвращается неинициализированной. Поддерживаются только числа и указатели.

**Параметры**

- L (lua_State*) – Lua_State
- ctypeid (uint32_t) – CTypeID из FFI для cdata

**Результат** область памяти, ассоциированная с cdata

См. также *luaL_checkcdata()*

void *luaL_checkcdata(struct lua_State *L, int idx, uint32_t *ctypeid)
Проверка, является ли аргумент функции idx cdata.

**Параметры**

- L (lua_State*) – Lua_State
- idx (int) – индекс стека

- `ctypeid` (`uint32_t*`) – аргумент вывода. CTypeID из FFI для возвращаемого cdata

**Результат** область памяти, ассоциированная с cdata

См. также *luaL_pushcdata()*

void `luaL_setcdatagc`(struct lua_State *L, int idx)
Определение функции-финализатора для cdata.

Аналог вызова `ffi.gc(obj, function)`. Функция-финализатор должна быть на вершине стека.

**Параметры**

- `L` (`lua_State*`) – Lua_State
- `idx` (`int`) – индекс стека

uint32_t `luaL_ctypeid`(struct lua_State *L, const char *ctypename)
Возврат CTypeID (FFI) заданного типа CDATA.

**Параметры**

- `L` (`lua_State*`) – Lua_State
- `ctypename` (`const char*`) – Имя типа в C в виде строки (например, «struct request» или «uint32_t»)

**Результат** CTypeID

См. также *luaL_pushcdata()*, *luaL_checkcdata()*

int `luaL_cdef`(struct lua_State *L, const char *ctypename)
Объявление символов для FFI.

**Параметры**

- `L` (`lua_State*`) – Lua_State
- `ctypename` (`const char*`) – C-определения (например, «struct stat»)

**Результат** 0, если выполнено

**Результат** `LUA_ERRRUN`, `LUA_ERRMEM` или `LUA_ERRERR`, в противном случае.

См. также `ffi.cdef(def)`

void `luaL_pushuint64`(struct lua_State *L, uint64_t val)
Принудительная передача uint64_t в стек.

**Параметры**

- `L` (`lua_State*`) – Lua_State
- `val` (`uint64_t`) – передаваемое значение

void `luaL_pushint64`(struct lua_State *L, int64_t val)
Принудительная передача int64_t в стек.

**Параметры**

- `L` (`lua_State*`) – Lua_State
- `val` (`int64_t`) – передаваемое значение

uint64_t `luaL_checkuint64`(struct lua_State *L, int idx)
Check whether the argument idx is a uint64 or a convertible string and returns this number.

**выбрасывает** ошибку, если аргумент нельзя конвертировать

uint64_t `luaL_checkint64`(struct lua_State *L, int idx)
Check whether the argument idx is a int64 or a convertible string and returns this number.

**выбрасывает** ошибку, если аргумент нельзя конвертировать

uint64_t `luaL_touint64`(struct lua_State *L, int idx)
Check whether the argument idx is a uint64 or a convertible string and returns this number.

**Результат** конвертированное число или 0, если аргумент нельзя конвертировать

int64_t `luaL_toint64`(struct lua_State *L, int idx)
Check whether the argument idx is a int64 or a convertible string and returns this number.

**Результат** конвертированное число или 0, если аргумент нельзя конвертировать

void `luaT_pushtuple`(struct lua_State *L, *box_tuple_t* *tuple)
Принудительная передача кортежа в стек.

**Параметры**

- `L` (`lua_State*`) – Lua_State

**выбрасывает** ошибка при нехватке памяти

См. также *luaT_istuple*

*box_tuple_t* *`luaT_istuple`(struct lua_State *L, int idx)
Проверка, является ли `idx` кортежем.

**Параметры**

- `L` (`lua_State*`) – Lua_State
- `idx` (`int`) – индекс стека

**Результат** не NULL, если `idx` – это кортеж

**Результат** NULL, если `idx` – это не кортеж

int `luaT_error`(lua_State *L)
Повторение последней ошибки в Tarantool в виде Lua-объекта.

**См. также** **lua_error()**, *box_error_last()*.

int `luaT_cpcall`(lua_State *L, lua_CFunction func, void *ud)
Аналог lua_cpcall(), но с соответствующей поддержкой ошибок Tarantool.

lua_State *`luaT_state`(void)
Получение глобального состояния Lua, используемого Tarantool.

### 9.7.10 Module say (logging)

enum `say_level`

enumerator ::`S_FATAL`
не используйте непосредственно данное значение

enumerator ::`S_SYSERROR`

enumerator ::`S_ERROR`

enumerator ::`S_CRIT`

enumerator ::`S_WARN`

enumerator ::S_INFO

enumerator ::S_VERBOSE

enumerator ::S_DEBUG

say(level, format, ...)

Форматирование и запись сообщения в файл журнала Tarantool.

**Параметры**

- level (int) – *log level*
- format (const char*) – строка в формате типа printf()
- ... – аргументы формата

См. также *printf(3)*, *say_level*

say_error(format, ...)

say_crit(format, ...)

say_warn(format, ...)

say_info(format, ...)

say_verbose(format, ...)

say_debug(format, ...)

say_syserror(format, ...)

Форматирование и запись сообщения в файл журнала Tarantool.

**Параметры**

- format (const char*) – строка в формате типа printf()
- ... – аргументы формата

См. также *printf(3)*, *say_level*

**Пример:**

```
say_info("Some useful information: %s", status);
```

### 9.7.11 Module schema

enum SCHEMA

enumerator ::BOX_SYSTEM_ID_MIN

Начало выделенного диапазона системных спейсов.

enumerator ::BOX_SCHEMA_ID

Идентификатор спейса _schema.

enumerator ::BOX_SPACE_ID

Идентификатор спейса _space.

enumerator ::BOX_VSPACE_ID
: Идентификатор виртуального спейса _vspace.

enumerator ::BOX_INDEX_ID
: Идентификатор спейса _index.

enumerator ::BOX_VINDEX_ID
: Идентификатор виртуального спейса _vindex.

enumerator ::BOX_FUNC_ID
: Идентификатор спейса _func.

enumerator ::BOX_VFUNC_ID
: Идентификатор виртуального спейса _vfunc.

enumerator ::BOX_USER_ID
: Идентификатор спейса _user.

enumerator ::BOX_VUSER_ID
: Идентификатор виртуального спейса _vuser.

enumerator ::BOX_PRIV_ID
: Идентификатор спейса _priv.

enumerator ::BOX_VPRIV_ID
: Идентификатор виртуального спейса _vpriv.

enumerator ::BOX_CLUSTER_ID
: Идентификатор спейса _cluster.

enumerator ::BOX_TRIGGER_ID
: Идентификатор спейса _trigger.

enumerator ::BOX_TRUNCATE_ID
: Идентификатор спейса _truncate.

enumerator ::BOX_SYSTEM_ID_MAX
: Окончание выделенного диапазона системных спейсов.

enumerator ::BOX_ID_NIL
: Нулевое значение NULL возвращается в случае ошибки.

### 9.7.12 Module trivia/config

API_EXPORT
: Внешний модификатор для всех доступных функций.

PACKAGE_VERSION_MAJOR
: Мажорная версия пакета – 1 в 2.0.5.

PACKAGE_VERSION_MINOR
: Минорная версия пакета – 0 в 2.0.5.

PACKAGE_VERSION_PATCH
: Патч-версия пакета – 5 в 2.0.5

PACKAGE_VERSION
: Строка с идентификатором версии: мажорная-минорная-патч-коммит-идентификатор, например, 2.0.5-75-gdd8e14ffb.

SYSCONF_DIR
: Директория для системной конфигурации (например, /etc)

`INSTALL_PREFIX`
: Префикс установки (например, `/usr`)

`BUILD_TYPE`
: Тип сборки, например, отладочная сборка или релиз.

`BUILD_INFO`
: Подпись типа сборки CMake, например, `Linux-x86_64-Debug`

`BUILD_OPTIONS`
: Командная строка для запуска CMake.

`COMPILER_INFO`
: Paths to C and CXX compilers.

`TARANTOOL_C_FLAGS`
: Флаги компиляции C, используемые для сборки Tarantool.

`TARANTOOL_CXX_FLAGS`
: Флаги компиляции CXX, используемые для сборки Tarantool.

`MODULE_LIBDIR`
: Путь для установки файлов модуля `*.lua`.

`MODULE_LUADIR`
: Путь для установки файлов модуля `*.so`/`*.dylib`

`MODULE_INCLUDEDIR`
: Путь к Lua (директория, где хранится этот файл).

`MODULE_LUAPATH`
: Постоянная, добавляемая к `package.path` в Lua для поиска файлов модуля `*.lua`.

`MODULE_LIBPATH`
: Постоянная, добавляемая к `package.cpath` в Lua для поиска файлов модуля `*.so`.

### 9.7.13 Module tuple

type `box_tuple_format_t`

*box_tuple_format_t* *`box_tuple_format_default`(void)
: Формат кортежа.

  Каждому кортежу соответствует определенный формат (класс). По умолчанию, используется формат для создания кортежей, не привязанных к определенному спейсу.

type `box_tuple_t`
: Кортеж

*box_tuple_t* *`box_tuple_new`(*box_tuple_format_t* *format, const char *tuple, const char *tuple_end)
: Выделение и инициализация нового кортежа из сырых данных MsgPack-массива.

  **Параметры**

  - `format` (*box_tuple_format_t**) – формат кортежа. Используйте *box_tuple_format_default()* для создания кортежа независимо от спейса.
  - `tuple` (`const char*`) – данные кортежа в формате MsgPack-массива ([ field1, field2, ...])
  - `tuple_end` (`const char*`) – конец данных `data`

  **Результат** NULL при нехватке памяти

**Результат** в остальных случаях кортеж

См. также *box.tuple.new()*

**Предупреждение:** При работе с кортежами в обязанности разработчика входит выделение достаточного места, уделяя особое внимание записи данных с помощью таких msgpuck-функций, как `mp_encode_array()`.

int `box_tuple_ref`(*box_tuple_t* *tuple)
Увеличение значения счетчика количества ссылок на кортеж.

Для кортежей подсчитываются ссылки. Все функции, которые возвращают кортежи, обеспечивают внутренний подсчет ссылок для последнего возвращенного кортежа до следующего вызова API-функции, которая передает управление или возвращает другой кортеж.

Следует увеличивать значение счетчика количества ссылок перед длительной обработкой кортежей в коде. Сборщик мусора в Lua не будет удалять кортежи с ссылками, даже если другой файбер удалит их из спейса. После обработки уменьшите значение счетчика количества ссылок с помощью *box_tuple_unref()*, иначе кортеж будет допускать утечку.

**Параметры**

- `tuple` (*box_tuple_t* *) – кортеж

**Результат** -1 в случае ошибки

**Результат** 0 в остальных случаях

См. также *box_tuple_unref()*

void `box_tuple_unref`(*box_tuple_t* *tuple)
Увеличение значения счетчика количества ссылок на кортеж.

**Параметры**

- `tuple` (*box_tuple_t* *) – кортеж

См. также *box_tuple_ref()*

uint32_t `box_tuple_field_count`(const *box_tuple_t* *tuple)
Возврат количества полей в кортеже (размер MsgPack-массива).

**Параметры**

- `tuple` (*box_tuple_t* *) – кортеж

size_t `box_tuple_bsize`(const *box_tuple_t* *tuple)
Возврат количества байтов, используемых для хранения внутренних данных кортежа (MsgPack-массив).

**Параметры**

- `tuple` (*box_tuple_t* *) – кортеж

ssize_t `box_tuple_to_buf`(const *box_tuple_t* *tuple, char *buf, size_t size)
Передача сырых MsgPack-данных в буфер памяти `buf` размера `size`.

Хранение полей кортежа в буфере памяти.

При успешном выполнении функция возвращает количество записанных байтов. Если размер буфера недостаточный, возвращается количество байтов, которое было бы записано, если бы было достаточно места.

**Результат** -1 в случае ошибки

**Результат** количество записанных байтов при успешном выполнении.

*box_tuple_format_t* *box_tuple_format(const *box_tuple_t* *tuple)
Возврат взаимосвязанного формата.

**Параметры**

- tuple (*box_tuple_t**) – кортеж

**Результат** формат кортежа

const char *box_tuple_field(const *box_tuple_t* *tuple, uint32_t field_id)
Возврат поля кортежа в MsgPack-формате. Результатом будет указатель на сырые данные в формате MessagePack, которые можно расшифровать с помощью функций mp_decode. Пример можно увидеть в программе практикума *read.c*.

Буфер действует до следующего вызова функции box_tuple_*.

**Параметры**

- tuple (*box_tuple_t**) – кортеж
- field_id (uint32_t) – индекс с основанием 0 в MsgPack-массиве.

**Результат** NULL, если i >= *box_tuple_field_count()*

**Результат** в остальных случаях msgpack

enum field_type

enumerator ::FIELD_TYPE_ANY

enumerator ::FIELD_TYPE_UNSIGNED

enumerator ::FIELD_TYPE_STRING

enumerator ::FIELD_TYPE_NUMBER

enumerator ::FIELD_TYPE_DOUBLE

enumerator ::FIELD_TYPE_INTEGER

enumerator ::FIELD_TYPE_BOOLEAN

enumerator ::FIELD_TYPE_VARBINARY

enumerator ::FIELD_TYPE_SCALAR

enumerator ::FIELD_TYPE_DECIMAL

enumerator ::FIELD_TYPE_ARRAY

enumerator ::FIELD_TYPE_MAX

Допустимые типы данных для полей кортежа.

Нельзя использовать макросы STRS/ENUM для типов, поскольку есть несоответствие между именем enum (STRING) и литералом имени типа («STR»). STR уже используется в качестве типа в Objective-C.

type box_key_def_t
Определение ключа

*box_key_def_t* *box_key_def_new(uint32_t *fields, uint32_t *types, uint32_t part_count)
Создание определения ключа с полям ключа с переданными типами по переданным позициям.

Можно использовать для создания формата кортежа и/или сопоставления кортежей.

**Параметры**

- `fields` (`uint32_t*`) – массив с идентификаторами поля ключа
- `types` (`uint32_t`) – массив с *типами поля* ключа
- `part_count` (`uint32_t`) – количество полей ключа

**Результат** определение ключа, если выполнено

**Результат** NULL в случае ошибки

void `box_key_def_delete`(*box_key_def_t* *key_def)
Удаление определения ключа

**Параметры**

- `key_def` (*box_key_def_t**) – удаляемое определение ключа

*box_tuple_format_t* *`box_tuple_format_new`(struct key_def *keys, uint16_t key_count)
Возврат нового формата кортежа на основании переданных определений ключа

**Параметры**

- `keys` (`key_def`) – массив ключей, определенный для формата
- `key_count` (`uint16_t`) – количество ключей

**Результат** новый формат кортежа, если выполнено

**Результат** NULL в случае ошибки

void `box_tuple_format_ref`(*box_tuple_format_t* *format)
Увеличение значения подсчета ссылок на формат кортежа

**Параметры**

- `tuple_format` (*box_tuple_format_t*) – формат кортежа для ссылок

void `box_tuple_format_unref`(*box_tuple_format_t* *format)
Уменьшение значения подсчета ссылок на формат кортежа

**Параметры**

- `tuple_format` (*box_tuple_format_t*) – формат кортежа для уменьшения

int `box_tuple_compare`(const *box_tuple_t* *tuple_a, const *box_tuple_t* *tuple_b, const *box_key_def_t* *key_def)
Сопоставление кортежей, используя определение ключа

**Параметры**

- `tuple_a` (`const` *box_tuple_t**) – первый кортеж
- `tuple_b` (`const` *box_tuple_t**) – второй кортеж
- `key_def` (`const` *box_key_def_t**) – определение ключа

**Результат** 0, если `key_fields(tuple_a)` == `key_fields(tuple_b)`

**Результат** <0, если `key_fields(tuple_a)` < `key_fields(tuple_b)`

**Результат** >0, если `key_fields(tuple_a)` > `key_fields(tuple_b)`

См. также enum *field_type*

int `box_tuple_compare_with_key`(const *box_tuple_t* *tuple, const char *key, const *box_key_def_t* *key_def);
Сопоставление кортежа с ключом, используя определение ключа

**Параметры**

- `tuple` (`const` *box_tuple_t*`*`) – кортеж
- `key` (`const char*`) – ключ с заголовком MessagePack-массива
- `key_def` (`const` *box_key_def_t*`*`) – определение ключа

**Результат** 0, если `key_fields(tuple)` == `parts(key)`

**Результат** <0, если `key_fields(tuple)` < `parts(key)`

**Результат** >0, если `key_fields(tuple)` > `parts(key)`

См. также enum *field_type*

type `box_tuple_iterator_t`
Итератор кортежей

*box_tuple_iterator_t* *`box_tuple_iterator(`*box_tuple_t* *`tuple)`
Выделение и инициализация нового итератора кортежей. Итератор кортежей позволяет проводить итерацию по полям на корневом уровне MsgPack-массива.

**Пример:**

```
box_tuple_iterator_t* it = box_tuple_iterator(tuple);
if (it == NULL) {
    // обработка ошибок с помощью box_error_last()
}
const char* field;
while (field = box_tuple_next(it)) {
    // обработка сырых MsgPack-данных
}

// перемотка итератора на начальное положение
box_tuple_rewind(it)
assert(box_tuple_position(it) == 0);

// перемотка на три поля
field = box_tuple_seek(it, 3);
assert(box_tuple_position(it) == 4);

box_iterator_free(it);
```

void `box_tuple_iterator_free(`*box_tuple_iterator_t* *`it)`
Удаление и освобождение итератора кортежей

uint32_t `box_tuple_position(`*box_tuple_iterator_t* *`it)`
Возврат следующего положения с основанием 0 в итераторе. То есть функция возвращает идентификатор поля, который вернется при следующем вызове *box_tuple_next()*. Возвращается значение 0 после инициализации или перемотки и *box_tuple_field_count()* по окончании итерации.

**Параметры**

- `it` (*box_tuple_iterator_t*`*`) – итератор кортежей

**Результат** положение

void `box_tuple_rewind(`*box_tuple_iterator_t* *`it)`
Перемотка итератора в начальное положение.

**Параметры**

- `it` (*box_tuple_iterator_t*`*`) – итератор кортежей

После: `box_tuple_position(it) == 0`

const char *`box_tuple_seek`(*box_tuple_iterator_t* *it, uint32_t field_no)
Поиск итератора кортежей.

Результатом будет указатель на сырые MessagePack-данные, которые можно расшифровать с помощью функций `mp_decode`. Пример можно увидеть в программе практикума *read.c*. Возвращаемый буфер действует до следующего вызова API `box_tuple_*` . Запрашиваемый номер поля `field_no` возвращается при следующем вызове `box_tuple_next(it)`.

**Параметры**

- `it` (*box_tuple_iterator_t*`*`) – итератор кортежей
- `field_no` (`uint32_t`) – номер поля – положение с основанием 0 в MsgPack-массиве

После:

- `box_tuple_position(it) == field_not`, если возвращается не NULL.
- `box_tuple_position(it) == box_tuple_field_count(tuple)`, если возвращается NULL.

const char *`box_tuple_next`(*box_tuple_iterator_t* *it)
Возврат следующего поля кортежа из итератора кортежей.

Результатом будет указатель на сырые MessagePack-данные, которые можно расшифровать с помощью функций mp_decode. Пример можно увидеть в программе практикума *read.c*. Возвращаемый буфер действует до следующего вызова API `box_tuple_*`.

**Параметры**

- `it` (*box_tuple_iterator_t*`*`) – итератор кортежей

**Результат** NULL, если полей больше нет

**Результат** в остальных случаях MsgPack

Ранее: *box_tuple_position()* – это идентификатор с основанием 0 возвращаемого поля.

После: `box_tuple_position(it) == box_tuple_field_count(tuple)`, если возвращается NULL.

*box_tuple_t* *`box_tuple_update`(const *box_tuple_t* *tuple, const char *expr, const char *expr_end)

*box_tuple_t* *`box_tuple_upsert`(const *box_tuple_t* *tuple, const char *expr, const char *expr_end)

### 9.7.14 Module txn

bool `box_txn`(void)
Возврат true (правда), если есть активная транзакция.

int `box_txn_begin`(void)
Начало транзакции в текущем файбере.

Транзакция привязана к вызывающему файберу, поэтому в одном файбере может быть только одна активная транзакция. См. также *box.begin()*.

**Результат** 0, если выполнено

**Результат** -1 в случае ошибки. Возможно, транзакция уже была запущена.

int `box_txn_commit`(void)
Коммит текущей транзакции. См. также *box.commit()*.

**Результат** 0, если выполнено

**Результат** -1 в случае ошибки. Возможен отказ записи на диск

void `box_txn_rollback`(void)
Откат текущей транзакции. См. также *box.rollback()*.

box_txn_savepoint_t *`savepoint`(void)
Возврат дескриптора контрольной точки.

void `box_txn_rollback_to_savepoint`(box_txn_savepoint_t *savepoint)
Откат текущей транзакции до указанной контрольной точки.

void *`box_txn_alloc`(size_t size)
Выделение памяти в пул памяти txn.

Память автоматически освобождается при коммите или откате транзакции.

**Результат** NULL при нехватке памяти

## 9.8 Детали реализации

### 9.8.1 Бинарный протокол

This section provides information on the Tarantool binary protocol, iproto. The protocol is called «binary» because the database is most frequently accessed via binary code instead of Lua request text. Tarantool experts use it:

- to write their own connectors
- to understand network messages
- to support new features that their favorite connector doesn't support yet
- to avoid repetitive parsing by the server

The binary protocol provides complete access to Tarantool functionality, including:

- мультиплексирование запросов, т.е. возможность асинхронной отправки множества запросов по одному соединению;
- формат ответа, который поддерживает запись в режиме без копирования (zero-copy).

---

**Примечание:** Since version *2.11.0*, you can use the *box.iproto* submodule to access IPROTO constants and features from Lua. The submodule enables to *send arbitrary IPROTO packets* over the session's socket and *override the behavior* for all IPROTO request types. Also, *IPROTO_UNKNOWN* constant is introduced. The constant is used for the *box.iproto.override()* API, which allows setting a handler for incoming requests with an unknown type.

---

### MP_* MessagePack types

The binary protocol handles data in the MessagePack format. Short descriptions of the basic MessagePack data types are on MessagePack's specification page. Tarantool also introduces several MessagePack type *extensions*.

In this document, MessagePack types are described by words that start with **MP_**. See this table:

| | |
|---|---|
| **MP_NIL** | nil |
| **MP_UINT** | unsigned integer |
| **MP_INT** | either integer or unsigned integer |
| **MP_STR** | string |
| **MP_BIN** | binary string |
| **MP_ARRAY** | array |
| **MP_MAP** | map |
| **MP_BOOL** | boolean |
| **MP_FLOAT** | float |
| **MP_DOUBLE** | double |
| **MP_EXT** | *extension* |
| **MP_OBJECT** | any MessagePack object |

### Request and response format

The types referred to in this document are MessagePack types. For their definitions, see the *MP_* MessagePack types* section.

### Packet structure

Requests and responses have similar structure. They contain three sections: size, header, and body.

It is legal to put more than one request in a packet.

#### Size

The size is an MP_UINT – unsigned integer, usually 32-bit. It is the size of the header plus the size of the body. It may be useful to compare it with the number of bytes remaining in the packet.

#### Header

The header is an MP_MAP. It may contain, in any order:

- Both the request and response use the *IPROTO_REQUEST_TYPE* key. It denotes the type of the packet.
- The request and the matching response have the same sync number (*IPROTO_SYNC*).
- *IPROTO_SCHEMA_VERSION* is an optional key that indicates whether there was a major change in the schema.
- In *interactive transactions*, every stream is identified by a unique *IPROTO_STREAM_ID*.

In case of replicating *synchronous transactions*, the header also contains the *IPROTO_FLAGS* key.

### Encoding and decoding

To see how Tarantool encodes the header, have a look at file xrow.c, function `xrow_header_encode`.

To see how Tarantool decodes the header, have a look at file net_box.c, function `netbox_decode_data`.

For example, in a successful response to `box.space:select()`, the IPROTO_REQUEST_TYPE value is 0 = `IPROTO_OK` and the array has all the tuples of the result.

Read the source code file net_box.c where the function `decode_metadata_optional` is an example of how Tarantool itself decodes extra items.

### Body

The body is an MP_MAP. Maximal iproto package body length is 2 GiB.

The body has the details of the request or response. In a request, it can also be absent or be an empty map. Both these states are interpreted equally. Responses contain the body anyway even for an *IPROTO_PING* request, where it is an empty MP_MAP.

A lot of responses contain the IPROTO_DATA map:

For most data-access requests (*IPROTO_SELECT*, *IPROTO_INSERT*, *IPROTO_DELETE*, etc.) the body is an IPROTO_DATA map with an array of tuples that contain an array of fields.

IPROTO_DATA is what we get with net_box and *Module buffer* so if we were using net_box we could decode with *msgpack.decode_unchecked()*, or we could convert to a string with `ffi.string(`*`pointer`*`,` *`length`*`)`. The *pickle.unpack()* function might also be helpful.

---

**Примечание:** For SQL-specific requests and responses, the body is a bit different. *Learn more* about this type of packets.

---

### Error responses

Instead of *IPROTO_OK*, an error response header has IPROTO_REQUEST_TYPE = *IPROTO_TYPE_ERROR*. Its code is `0x8XXX`, where `XXX` is the error code – a value in src/box/errcode.h. `src/box/errcode.h` also has some convenience macros which define hexadecimal constants for return codes.

The error response body is a map that contains two keys: *IPROTO_ERROR* and *IPROTO_ERROR_24*. While IPROTO_ERROR contains an MP_MAP value, IPROTO_ERROR_24 contains a string. The two keys are provided to accommodate clients with older and newer Tarantool versions.

### Error responses before 2.4.1

Before Tarantool v. *2.4.1*, the key IPROTO_ERROR contained a string and was identical to the current IPROTO_ERROR_24 key.

Let's consider an example. This is the fifth message, and the request was to create a duplicate space with `conn:eval([[box.schema.space.create('_space');]])`. The unsuccessful response looks like this:

The tutorial *Understanding the binary protocol* shows actual byte codes of the response to the IPROTO_EVAL message.

Looking in errcode.h, we find that the error code `0x0a` (decimal 10) is ER_SPACE_EXISTS, and the string associated with ER_SPACE_EXISTS is «Space „%s“ already exists».

Since version *2.4.1*, responses for errors have extra information following what was described above. This extra information is given via the MP_ERROR extension type. See details in the *MessagePack extensions* section.

### Keys used in requests and responses

This section describes `iproto` keys contained in requests and responses. The keys are Tarantool constants that are either defined or mentioned in the iproto_constants.h file.

While the keys themselves are unsigned 8-bit integers, their values can have different types.

### Basic description

### General

| Name | Code and value type | Description |
|---|---|---|
| *IPROTO_VERSION* | 0x54 MP_UINT | Binary protocol version supported by the client |
| *IPROTO_FEATURES* | 0x55 MP_ARRAY | Supported binary protocol features |
| *IPROTO_SYNC* | 0x01 MP_UINT | Unique request identifier |
| *IPROTO_SCHEMA_VERSION* | 0x05 MP_UINT | Version of the database schema |
| IPROTO_TIMESTAMP | 0x04 MP_DOUBLE | Time in seconds since the Unix epoch |
| *IPROTO_REQUEST_TYPE* | 0x00 MP_UINT | Request type or response type |
| *IPROTO_ERROR* | 0x52 *MP_ERROR* | Error response |
| *IPROTO_ERROR_24* | 0x31 MP_STR | Error as a string |
| IPROTO_DATA | 0x30 MP_OBJECT | Data passed in the transaction. Can be empty. Used in all requests and responses |
| IPROTO_SPACE_ID | 0x10 MP_UINT | Space identifier |
| IPROTO_INDEX_ID | 0x11 MP_UINT | Index identifier |
| *IPROTO_TUPLE* | 0x21 MP_ARRAY | Tuple, arguments, operations, or authentication pair. *See details* |
| IPROTO_KEY | 0x20 MP_ARRAY | Array of index keys in the request. See *space_object:select()* |
| IPROTO_LIMIT | 0x12 MP_UINT | Maximum number of tuples in the space |
| IPROTO_OFFSET | 0x13 MP_UINT | Number of tuples to skip in the select |
| *IPROTO_ITERATOR* | 0x14 MP_UINT | Iterator type |
| IPROTO_INDEX_BASE | 0x15 MP_UINT | Indicates whether the first field number is 1 or 0 |
| IPROTO_FUNCTION_NAME | 0x22 MP_STR | Name of the called function. Used in *IPROTO_CALL* |
| IPROTO_USER_NAME | 0x23 MP_STR | User name. Used in *IPROTO_AUTH* |
| IPROTO_OPS | 0x28 MP_ARRAY | Array of operations. Used in *IPROTO_UPSERT* |
| IPROTO_EXPR | 0x27 MP_STR | Command argument. Used in *IPROTO_EVAL* |
| IPROTO_AUTH_TYPE | 0x5b MP_STR | A *protocol* used to generate user authentication data |
| IPROTO_AFTER_POSITION | 0x2e MP_STR | The position of a tuple after which *space_object:select()* starts the search |
| IPROTO_AFTER_TUPLE | 0x2f MP_ARRAY | A tuple after which *space_object:select()* starts the search |
| IPROTO_FETCH_POSITION | 0x1f MP_BOOL | If **true**, *space_object:select()* returns the position of the last selected tuple |
| IPROTO_POSITION | 0x35 MP_STR | If `IPROTO_FETCH_POSITION` is **true**, returns a base64-encoded string representing the position of the last selected tuple |

### Streams

| Name | Code and value type | Description |
|---|---|---|
| *IPROTO_STREAM_ID* | 0x0a MP_UINT | Unique *stream* identifier |
| IPROTO_TIMEOUT | 0x56 MP_DOUBLE | Timeout in seconds, after which the transactions are rolled back |
| *IPROTO_TXN_ISOLATION* | 0x59 MP_UINT | Transaction isolation level |

### General replication

| Name | Code and value type | Description |
|---|---|---|
| IPROTO_REPLICA_ID | 0x02 MP_INT | Replica ID |
| IPROTO_INSTANCE_UUID | 0x24 MP_STR | Instance UUID |
| *IPROTO_VCLOCK* | 0x26 MP_MAP | The instance's vclock |
| *IPROTO_VCLOCK_SYNC* | 0x5a MP_UINT | ID of the vclock synchronization request. Since 2.11 |
| IPROTO_REPLICASET_UUID | 0x25 MP_STR | Before Tarantool version 2.11, IPROTO_REPLICASET_UUID was called IPROTO_CLUSTER_UUID. |
| IPROTO_LSN | 0x03 MP_UINT | Log sequence number of the transaction |
| IPROTO_TSN | 0x08 MP_UINT | Transaction sequence number |
| *IPROTO_BALLOT_IS_RO_CFG* | 0x01 MP_BOOL | True if the instance is configured as *read_only*. Since *2.6.1* |
| *IPROTO_BALLOT_VCLOCK* | 0x02 MP_MAP | Current vclock of the instance |
| *IPROTO_BALLOT_GC_VCLOCK* | 0x03 MP_MAP | Vclock of the instance's oldest WAL entry |
| *IPROTO_BALLOT_IS_RO* | 0x04 MP_BOOL | True if the instance is not writable: configured as *read_only*, has *orphan status*, or is a *Raft follower*. Since *2.6.1* |
| *IPROTO_BALLOT_IS_ANON* | 0x05 MP_BOOL | True if the replica is anonymous. Corresponds to *box.cfg.replication_anon*. Since *2.7.1* |
| *IPROTO_BALLOT_IS_BOOTED* | 0x06 MP_BOOL | True if the instance has finished its bootstrap or recovery process. Since *2.7.3*, *2.8.2*, *2.10.0* |
| *IPROTO_BALLOT_CAN_LEAD* | 0x07 MP_BOOL | True if *box.cfg.election_mode* is `candidate` or `manual`. Since v. *2.7.3* and *2.8.2* |
| *IPROTO_BALLOT_BOOTSTRAP_LEADER_UUID* | 0x08 MP_STR | UUID of the bootstrap leader. The UUID is encoded as a 36-byte string. Since v. 2.11 |
| *IPROTO_BALLOT_REGISTERED_REPLICA_UUIDS* | 0x09 MP_ARRAY | An array of MP_STR elements that contains the UUIDs of members registered in the replica set. Each UUID is encoded as a 36-byte string. Since v. 2.11 |
| *IPROTO_FLAGS* | 0x09 MP_UINT | Auxiliary data to indicate the last transaction message state. Included in the header of any DML request that is recorded in the WAL. |
| IPROTO_SERVER_VERSION | 0x06 MP_UINT | Tarantool version of the subscribing node, in a compact representation |
| IPROTO_REPLICA_ANON | 0x50 MP_BOOL | Optional key used in *SUBSCRIBE request*. True if the subscribing replica is anonymous |
| IPROTO_ID_FILTER | 0x51 MP_ARRAY | Optional key used in *SUBSCRIBE request*, followed by an array of ids of instances whose rows won't be relayed to the replica. Since v. *2.10.0* |

### Synchronous replication

| Name | Code and value type | Description |
|---|---|---|
| *IPROTO_TERM* | 0x53 MP_UINT | *RAFT term* on an instance |
| IPROTO_RAFT_TERM | 0x00 MP_UINT | *RAFT term* on an instance. The key is only used for requests with the *IPROTO_RAFT* type. |
| IPROTO_RAFT_VOTE | 0x01 MP_UINT | Instance vote in the current term (if any) |
| IPROTO_RAFT_STATE | 0x02 MP_UINT | *RAFT state*. Possible values: 1 – follower, 2 – candidate, 3 – leader |
| *IPROTO_RAFT_VCLOCK* | 0x03 MP_MAP | Current vclock of the instance |
| IPROTO_RAFT_LEADER_ID | 0x04 MP_UINT | Current leader node ID as seen by the node that issues the request Since version *2.10.0* |
| IPROTO_RAFT_IS_LEADER_SEEN | 0x05 MP_BOOL | True if the node has a direct connection to the leader node. Since version *2.10.0* |

All `IPROTO_RAFT_*` keys are used only in `IPROTO_RAFT*` requests.

### Events and subscriptions

| Name | Code and value type | Description |
|---|---|---|
| IPROTO_EVENT_KEY | 0x56 MP_STR | *Event* key name |
| IPROTO_EVENT_DATA | 0x57 MP_OBJECT | *Event* data sent to a remote watcher |

*Learn more about events and subscriptions in iproto.*

### SQL-specific

These keys are used with SQL within *SQL-specific requests and responses* like *IPROTO_EXECUTE* and *IPROTO_PREPARE*.

| Name | Code and value type | Description |
|---|---|---|
| IPROTO_SQL_TEXT | 0x40 MP_STR | SQL statement text |
| IPROTO_STMT_ID | 0x43 MP_INT | Identifier of the prepared statement |
| IPROTO_OPTIONS | 0x2b MP_ARRAY | SQL transaction options. Usually empty |
| *IPROTO_METADATA* | 0x32 MP_ARRAY of MP_MAP items | SQL transaction metadata |
| IPROTO_FIELD_NAME | 0x00 MP_STR | Field name. Nested in *IPROTO_METADATA* |
| IPROTO_FIELD_TYPE | 0x01 MP_STR | Field type. Nested in *IPROTO_METADATA* |
| IPROTO_FIELD_COLL | 0x02 MP_STR | Field collation. Nested in *IPROTO_METADATA* |
| IPROTO_FIELD_IS_NULLABLE | 0x03 MP_BOOL | True if the field is nullable. Nested in *IPROTO_METADATA*. |
| IPROTO_FIELD_IS_AUTOINCREMENT | 0x04 MP_BOOL | True if the field is auto-incremented. Nested in *IPROTO_METADATA*. |
| IPROTO_FIELD_SPAN | 0x05 MP_STR or MP_NIL | Original expression under SELECT. Nested in *IPROTO_METADATA*. See *box.execute()* |
| IPROTO_BIND_METADATA | 0x33 MP_ARRAY | Bind variable names and types |
| IPROTO_BIND_COUNT | 0x34 MP_INT | Number of parameters to bind |
| *IPROTO_SQL_BIND* | 0x41 MP_ARRAY | Parameter values to match ? placeholders or :name placeholders |
| IPROTO_SQL_INFO | 0x42 MP_MAP | Additional SQL-related parameters |
| SQL_INFO_ROW_COUNT | 0x00 MP_UINT | Number of changed rows. Is `0` for statements that do not change rows. Nested in IPROTO_SQL_INFO |
| SQL_INFO_AUTO_INCREMENT_IDS | 0x01 MP_ARRAY of MP_UINT items | New primary key value (or values) for an INSERT in a table defined with PRIMARY KEY AUTOINCREMENT. Nested in IPROTO_SQL_INFO |

### Details on individual keys

#### IPROTO_VERSION

Code: 0x54.

IPROTO_VERSION is an integer number reflecting the version of protocol that the client supports. The latest IPROTO_VERSION is 3.

### IPROTO_FEATURES

Code: 0x55.

Available IPROTO_FEATURES are the following:

- `IPROTO_FEATURE_STREAMS = 0` – streams support: *IPROTO_STREAM_ID* in the request header.
- `IPROTO_FEATURE_TRANSACTIONS = 1` – transaction support: IPROTO_BEGIN, IPROTO_COMMIT, and IPROTO_ROLLBACK commands (with *IPROTO_STREAM_ID* in the request header). Learn more about *sending transaction commands*.
- `IPROTO_FEATURE_ERROR_EXTENSION = 2` – *MP_ERROR* MsgPack extension support. Clients that don't support this feature receive error responses for *IPROTO_EVAL* and *IPROTO_CALL* encoded to string error messages.
- `IPROTO_FEATURE_WATCHERS = 3` – remote watchers support: *IPROTO_WATCH*, *IPROTO_UNWATCH*, and *IPROTO_EVENT* commands.

### IPROTO_SYNC

Code: 0x01.

This is an unsigned integer that should be incremented so that it is unique in every request. This integer is also returned from *box.session.sync()*.

The IPROTO_SYNC value of a response should be the same as the IPROTO_SYNC value of a request.

### IPROTO_SCHEMA_VERSION

Code: 0x05.

Version of the database schema – an unsigned number that goes up when there is a major change in the schema.

In a *request* header, IPROTO_SCHEMA_VERSION is optional, so the version will not be checked if it is absent.

In a *response* header, IPROTO_SCHEMA_VERSION is always present, and it is up to the client to check if it has changed.

### IPROTO_ITERATOR

Code: 0x14.

Possible values (see iterator_type.h):

| | |
|---|---|
| 0 | *EQ* |
| 1 | *REQ* |
| 2 | *ALL*, all tuples |
| 3 | *LT*, less than |
| 4 | *LE*, less than or equal |
| 5 | *GE*, greater than or equal |
| 6 | *GT*, greater than |
| 7 | *BITS_ALL_SET*, all bits of the value are set in the key |
| 8 | *BITS_ANY_SET*, at least one bit of the value is set |
| 9 | *BITS_ALL_NOT_SET*, no bits are set |
| 10 | *OVERLAPS*, overlaps the rectangle or box |
| 11 | *NEIGHBOR*, neighbors the rectangle or box |

### IPROTO_STREAM_ID

Code: 0x0a.

Only used in *streams*. This is an unsigned number that should be unique in every stream.

In requests, IPROTO_STREAM_ID is useful for two things: ensuring that requests within transactions are done in separate groups, and ensuring strictly consistent execution of requests (whether or not they are within transactions).

In responses, IPROTO_STREAM_ID does not appear.

See *Binary protocol – streams*.

### IPROTO_TXN_ISOLATION

IPROTO_TXN_ISOLATION is the *transaction isolation level*. It can take the following values:

- `TXN_ISOLATION_DEFAULT = 0` – use the default level from `box.cfg` (default value)
- `TXN_ISOLATION_READ_COMMITTED = 1` – read changes that are committed but not confirmed yet
- `TXN_ISOLATION_READ_CONFIRMED = 2` – read confirmed changes
- `TXN_ISOLATION_BEST_EFFORT = 3` – determine isolation level automatically

See *Binary protocol – streams* to learn more about stream transactions in the binary protocol.

### IPROTO_REQUEST_TYPE

Code: 0x00.

The key is used both in requests and responses. It indicates the request or response type and has any request or response name for the value (example: IPROTO_AUTH). See requests and responses for *client-server communication*, *replication*, *events and subscriptions*, *streams and interactive transactions*.

### IPROTO_ERROR

Code: 0x52.

In case of error, the response body contains IPROTO_ERROR and *IPROTO_ERROR_24* instead of IPROTO_DATA.

To learn more about error responses, check the section *Request and response format*.

### IPROTO_ERROR_24

Code: 0x31.

IPROTO_ERROR_24 is used in Tarantool versions before *2.4.1*. The key contains the error in the string format.

Since *Tarantool 2.4.1*, Tarantool packs errors as the *MP_ERROR* MessagePack extension, which includes extra information. Two keys are passed in the error response body: IPROTO_ERROR and IPROTO_ERROR_24.

To learn more about error responses, check the section *Request and response format*.

### IPROTO_TUPLE

Code: 0x21.

Multiple operations make use of this key in different ways:

| | |
|---|---|
| *IPROTO_INSERT*, *IPROTO_REPLACE*, *IPROTO_UPSERT* | Tuple to be inserted |
| *IPROTO_UPSERT* | Operations to perform |
| *IPROTO_AUTH* | Array of 2 fields: authentication mechanism and scramble, encrypted according to the specified mechanism. See more on the *authentication* sequence. |
| *IPROTO_CALL*, *IPROTO_EVAL* | Array of arguments |

### IPROTO_FLAGS

Code: 0x09.

When it comes to replicating synchronous transactions, the IPROTO_FLAGS key is included in the header. The key contains an MP_UINT value of one or more bits:

- IPROTO_FLAG_COMMIT (0x01) is set if this is the last message for a transaction.
- IPROTO_FLAG_WAIT_SYNC (0x02) is set if this is the last message for a transaction which cannot be completed immediately.
- IPROTO_FLAG_WAIT_ACK (0x04) is set if this is the last message for a synchronous transaction.

Example:

### IPROTO_TERM

Code: 0x53.

- The key is used in the *IPROTO_RAFT_PROMOTE* and *IPROTO_RAFT_DEMOTE* requests.
- Since version 2.11, the key is included in response to a *heartbeat message*. The term corresponds to the value of *box.info.synchro.queue.term* on the sender instance.

#### Vclock keys

The vclock (vector clock) is a log sequence number map that defines the version of the dataset stored on the node. In fact, it represents the number of logical operations executed on a specific node. A vclock looks like this:

There are five keys that correspond to vector clocks in different contexts of replication. They all have the MP_MAP type:

- IPROTO_VCLOCK (0x26) is passed to a new instance *joining the replica set*.
- IPROTO_VCLOCK_SYNC (0x5a) is used by *replication heartbeats*. The master sends its heartbeats, including this monotonically growing key, to a replica. Once the replica receives a heartbeat with a non-zero IPROTO_VCLOCK_SYNC value, it starts responding with the same value in all its acknowledgements. This key was introduced in version 2.11.
- IPROTO_BALLOT_VCLOCK (0x02) is included in the *IPROTO_BALLOT* message. IPROTO_BALLOT is sent in response to the *IPROTO_VOTE* request. This key was introduced in *Tarantool 2.6.1*.
- IPROTO_BALLOT_GC_VCLOCK (0x03) is also included in the *IPROTO_BALLOT* message. IPROTO_BALLOT is sent in response to the *IPROTO_VOTE* request. It is the vclock of the oldest WAL entry on the instance. Corresponds to *box.info.gc().vclock*. This key was introduced in *Tarantool 2.6.1*.
- IPROTO_RAFT_VCLOCK (0x03) is included in the *IPROTO_RAFT* message. It is present only on the instances in the *«candidate» state* (IPROTO_RAFT_STATE == 2).

#### IPROTO_BALLOT keys

All IPROTO_BALLOT_* keys are only used in the *IPROTO_BALLOT* requests. There have been the following name changes starting with versions *Tarantool 2.7.3*, *Tarantool 2.8.2*, and *Tarantool 2.10.0*:

- IPROTO_BALLOT_IS_RO_CFG (0x01) was formerly called IPROTO_BALLOT_IS_RO.
- IPROTO_BALLOT_IS_RO (0x04) was formerly called IPROTO_BALLOT_IS_LOADING.

### IPROTO_METADATA

Code: 0x32.

Used with SQL within *IPROTO_EXECUTE*.

The key contains an array of column maps, with each column map containing at least IPROTO_FIELD_NAME (0x00) and MP_STR, and IPROTO_FIELD_TYPE (0x01) and MP_STR.

Additionally, if `sql_full_metadata` in the *_session_settings* system space is TRUE, then the array has these additional column maps which correspond to components described in the *box.execute()* section.

### IPROTO_SQL_BIND

Code: 0x41.

Used with SQL within *IPROTO_EXECUTE*.

IPROTO_SQL_BIND is an array of parameter values to match ? placeholders or :name placeholders. It can contain values of any type, including MP_MAP.

- Values that are not MP_MAP replace the `?` placeholders in the request.
- MP_MAP values must have the format `{[name] = value}`, where `name` is the named parameter in the request. Here is an example of such a request:

```
tarantool> conn:execute('SELECT ?, ?, :name1, ?, :name2, :name1', {1, 2, {[':name1
↪'] = 5}, 'str', {[':name2'] = true}})
---
- metadata:
- name: COLUMN_1
    type: integer
- name: COLUMN_2
    type: integer
- name: COLUMN_3
    type: integer
- name: COLUMN_4
    type: text
- name: COLUMN_5
    type: boolean
- name: COLUMN_6
    type: boolean
rows:
- [1, 2, 5, 'str', true, 5]
```

#### Client-server requests and responses

This section describes client requests, their arguments, and the values returned by the server.

Some requests are described on separate pages. Those are the requests related to:

- *stream transactions*
- *asynchronous server-client notifications*
- *replication*
- *SQL* – *IPROTO_EXECUTE* and *IPROTO_PREPARE*.

### Overview

| Name | Code | Description |
|---|---|---|
| *IPROTO_OK* | 0x00 MP_UINT | Successful response |
| *IPROTO_CHUNK* | 0x80 MP_UINT | Out-of-band response |
| *IPROTO_TYPE_ERROR* | 0x8XXX MP_INT | Error response |
| *IPROTO_UNKNOWN* | -1 MP_UINT | An unknown request type |
| *IPROTO_SELECT* | 0x01 | *Select* request |
| *IPROTO_INSERT* | 0x02 | *Insert* request |
| *IPROTO_REPLACE* | 0x03 | *Replace* request |
| *IPROTO_UPDATE* | 0x04 | *Update* request |
| *IPROTO_UPSERT* | 0x09 | *Upsert* request |
| *IPROTO_DELETE* | 0x05 | *Delete* request |
| *IPROTO_CALL* | 0x0a | Function remote call (*conn:call()*) |
| *IPROTO_AUTH* | 0x07 | Authentication request |
| *IPROTO_EVAL* | 0x08 | Evaluate a Lua expression (*conn:eval()*) |
| *IPROTO_NOP* | 0x0c | Increment the LSN and do nothing else |
| *IPROTO_PING* | 0x40 | Ping (*conn:ping()*) |
| *IPROTO_ID* | 0x49 | Share iproto version and supported features |

#### IPROTO_OK

Code: 0x00.

This request/response type is contained in the header and signifies success. Here is an example:

#### IPROTO_CHUNK

Code: 0x80.

If the response is out-of-band, due to use of *box.session.push()*, then IPROTO_REQUEST_TYPE is IPROTO_CHUNK instead of IPROTO_OK.

#### IPROTO_TYPE_ERROR

Code: 0x8XXX (see below).

Instead of *IPROTO_OK*, an error response header has `0x8XXX` for IPROTO_REQUEST_TYPE. `XXX` is the error code – a value in src/box/errcode.h. `src/box/errcode.h` also has some convenience macros which define hexadecimal constants for return codes.

To learn more about error responses, check the section *Request and response format*.

### IPROTO_UNKNOWN

Since *2.11.0*.

Code: -1.

An unknown request type. The constant is used to override the handler of unknown IPROTO request types. Learn more: *box.iproto.override()* and *box_iproto_override*.

### IPROTO_SELECT

Code: 0x01.

См. *space_object:select()*. Тело сообщения представляет собой ассоциативный массив из 6 элементов.

#### Пример

Если ID спейса `tspace` = 512 и это пятое сообщение, *conn*`.space.tspace:select({0},{iterator='GT',offset=1,limit=2})` вызовет следующий пакет запроса:

Байт-коды сообщения IPROTO_SELECT рассмотрены в разделе *Примеры*

### IPROTO_INSERT

Code: 0x02.

См. *space_object:insert()*. Тело сообщения представляет собой ассоциативный массив из 2 элементов:

For example, if the request is `INSERT INTO` *`table-name`* `VALUES (1), (2), (3)`, then the response body contains an `IPROTO_SQL_INFO` map with `SQL_INFO_ROW_COUNT = 3`. `SQL_INFO_ROW_COUNT` can be 0 for statements that do not change rows, but can be 1 for statements that create new objects.

#### Пример

Если ID спейса `tspace` = 512 и это пятое сообщение, *conn*`.space.tspace:insert{1, 'AAA'}` вызовет следующие пакеты запроса и ответа:

The tutorial *Understanding the binary protocol* shows actual byte codes of the response to the IPROTO_INSERT message.

### IPROTO_REPLACE

Code: 0x03.

См. *space_object:replace()*. Тело сообщения представляет собой ассоциативный массив из 2 элементов, как и в случае IPROTO_INSERT:

### IPROTO_UPDATE

Code: 0x04.

См. *space_object:update()*.

Тело сообщения обычно представляет собой ассоциативный массив из 4 элементов:

#### Examples

If the operation specifies no values, then IPROTO_TUPLE is a 2-item array:

Normally field numbers start with 1.

If the operation specifies one value, then IPROTO_TUPLE is a 3-item array:

Otherwise IPROTO_TUPLE is a 5-item array:

Если ID спейса `tspace` = 512 и это пятое сообщение, *conn*`.space.tspace:update(999, {{'=', 2, 'B'}})` вызовет следующий пакет запроса:

The map item IPROTO_INDEX_BASE is optional.

The tutorial *Understanding the binary protocol* shows the actual byte codes of an IPROTO_UPDATE message.

### IPROTO_UPSERT

Code: 0x09.

См. *space_object:upsert()*.

Тело сообщения обычно представляет собой ассоциативный массив из 4 элементов:

IPROTO_OPS is the array of operations. It is the same as the IPROTO_TUPLE of *IPROTO_UPDATE*.

IPROTO_TUPLE is an array of primary-key field values.

### IPROTO_DELETE

Code: 0x05.

См. *space_object:delete()*. Тело сообщения представляет собой ассоциативный массив из 3 элементов:

### IPROTO_EVAL

Code: 0x08.

See *conn:eval()*. Since the argument is a Lua expression, this is Tarantool's way to handle non-binary with the binary protocol. Any request that does not have its own code, for example `box.space.`*space-name*`:drop()`, will be handled either with *IPROTO_CALL* or IPROTO_EVAL.

The *tt* administrative utility makes extensive use of `eval`.

The body is a 2-item map:

- For *IPROTO_EVAL* and *IPROTO_CALL* the response body will usually be an array but, since Lua requests can result in a wide variety of structures, bodies can have a wide variety of structures.

**Примечание:** For SQL-specific responses, the body is a bit different. *Learn more* about this type of packets.

#### Пример

Если это пятое сообщение, `conn:eval('return 5;')` приведет к следующему:

### IPROTO_CALL

Code: 0x0a.

See *conn:call()*. This is a remote stored-procedure call. *Tarantool 1.6* and earlier made use of the IPROTO_CALL_16 request (code: 0x06). It is now deprecated and superseded by IPROTO_CALL.

The body is a 2-item map. The response will be a list of values, similar to the *IPROTO_EVAL* response. The return from conn:call is whatever the function returns.

**Примечание:** For SQL-specific responses, the body is a bit different. *Learn more* about this type of packets.

### IPROTO_AUTH

Code: 0x07.

For general information, see the *Access control* section in the administrator's guide.

For more on how authentication is handled in the binary protocol, see the *Authentication* section of this document.

The client sends an authentication packet as an IPROTO_AUTH message:

IPROTO_USERNAME holds the user name. IPROTO_TUPLE must be an array of 2 fields: authentication mechanism and scramble, encrypted according to the specified mechanism.

The server instance responds to an authentication packet with a standard response with 0 tuples.

To see how Tarantool handles this, look at net_box.c function `netbox_encode_auth`.

### IPROTO_NOP

Code: 0x0c.

Нет такого запроса на Lua, который бы соответствовал IPROTO_NOP. IPROTO_NOP приводит к увеличению LSN. Иногда константу можно использовать для обновления значения, когда старое и новое значения одинаковы, но LSN нужно увеличить, поскольку нужно зарегистрировать изменение данных. Тело сообщения пустое.

### IPROTO_PING

Code: 0x40.

См. *conn:ping()*. В теле сообщения будет пустой ассоциативный массив, потому что IPROTO_PING в заголовке содержит всю информацию, необходимую экземпляру сервера.

### IPROTO_ID

Code: 0x49.

Clients send this message to inform the server about the protocol version and features they support. Based on this information, the server can enable or disable certain features in interacting with these clients.

The body is a 2-item map:

The response body has the same structure as the request body. It informs the client about the protocol version, features supported by the server, and a protocol used to generate user authentication data.

IPROTO_ID requests can be processed without authentication.

#### Session start and authentication

Every iproto session begins with a greeting and optional authentication.

#### Приветствие

Когда клиент подключается к экземпляру сервера, тот выдает в ответе 128-байтовое текстовое сообщение приветствия не в формате MsgPack:

```
Tarantool <version> (<protocol>) <instance-uuid>
<salt>
```

Пример:

```
Tarantool 2.10.0 (Binary) 29b74bed-fdc5-454c-a828-1d4bf42c639a
QK2HoFZGXTXBq2vFj7soCsHqTo6PGTF575ssUBAJLAI=
```

Приветствие содержит две 64-байтные строки текста в формате ASCII. Каждая строка заканчивается символом разрыва строки (\n). Если длина содержимого строки менее 64 байтов, то оставшееся место заполняется символами с ASCII-кодом 0. В консоли эти символы не отображаются.

The first line contains the instance version and protocol type. The second line contains the session salt – a base64-encoded random string, which is usually 44 bytes long. The salt is used in the authentication packet – the *IPROTO_AUTH message*.

#### Аутентификация

If authentication is skipped, then the session user is `'guest'` (the `'guest'` user does not need a password).

If authentication is not skipped, then at any time an *authentication packet* can be prepared using the greeting, the user's name and password, and sha-1 functions, as follows.

```
PREPARE SCRAMBLE:

    size_of_encoded_salt_in_greeting = 44;
    size_of_salt_after_base64_decode = 32;
     /* sha1() will only use the first 20 bytes */
    size_of_any_sha1_digest = 20;
    size_of_scramble = 20;

prepare 'chap-sha1' scramble:

    salt = base64_decode(encoded_salt);
    step_1 = sha1(password);
    step_2 = sha1(step_1);
    step_3 = sha1(first_20_bytes_of_salt, step_2);
    scramble = xor(step_1, step_3);
    return scramble;
```

### Streams

#### Overview

The *Streams and interactive transactions* feature, which was added in Tarantool version v. 2.10.0, allows two things: sequential processing and interleaving.

**Sequential processing**: With streams there is a guarantee that the server instance will not handle the next request in a stream until it has completed the previous one.

**Interleaving**: For example, a series of requests can include «begin for stream #1», «begin for stream #2», «insert for stream #1», «insert for stream #2», «delete for stream #1», «commit for stream #1», «rollback for stream #2».

To work with stream transactions using iproto, the following is required:

- The engine should be *vinyl* or *memtx with mvcc*.
- The client is responsible for ensuring that the stream identifier, unsigned integer *IPROTO_STREAM_ID*, is in the request header. IPROTO_STREAM_ID can be any positive 64-bit number, and should be unique for the connection. If IPROTO_STREAM_ID equals zero, the server instance will ignore it.

### Basic request description

| Name | Code | Description |
|---|---|---|
| *IPROTO_BEGIN* | 0x0e | Begin a transaction in the specified stream |
| *IPROTO_COMMIT* | 0x0f | Commit the transaction in the specified stream |
| *IPROTO_ROLLBACK* | 0x10 | Rollback the transaction in the specified stream |

### IPROTO_BEGIN

Code: 0x0e.

Begin a transaction in the specified stream. See *stream:begin()*. The body is optional and can contain two items:

IPROTO_TIMEOUT is an optional timeout (in seconds). After it expires, the transaction will be rolled back automatically.

### IPROTO_COMMIT

Code: 0x0f.

Commit the transaction in the specified stream. See *stream:commit()*.

### IPROTO_ROLLBACK

Codde: 0x10.

Rollback the transaction in the specified stream. See *stream:rollback()*.

### Example

Suppose that the client has started a stream with the *net.box module*

```
net_box = require('net.box')
conn = net_box.connect('localhost:3302')
stream = conn:new_stream()
```

В этот момент объект `stream` будет выглядеть так же, как объект `conn`, но включать один дополнительный элемент: `stream_id`. Пусть теперь клиент, используя `stream` вместо `conn`, отправит два запроса:

```
stream.space.T:insert{1}
stream.space.T:insert{2}
```

Заголовок и тело этих запросов будут такими же, как в обычных запросах *IPROTO_INSERT*, но заголовок будет содержать дополнительный элемент `IPROTO_STREAM_ID=0x0a`, где `MP_UINT=0x01`. В этом примере значение IPROTO_STREAM_ID равно 1, так как при вызове `conn:new_stream()` идентификатору каждого нового стрима присваивается уникальное значение, начиная с 1.

The client makes stream transactions by sending, in order:

1. IPROTO_BEGIN with an optional transaction timeout in the IPROTO_TIMEOUT field of the request body.
2. The transaction data-change and query requests.
3. IPROTO_COMMIT or IPROTO_ROLLBACK.

All these requests must contain the same IPROTO_STREAM_ID value.

A rollback will happen automatically if a disconnect occurs or the transaction timeout expires before the commit is possible.

Thus there are now multiple ways to do transactions: with `net_box` `stream:begin()` and `stream:commit()` or `stream:rollback()` which cause IPROTO_BEGIN and IPROTO_COMMIT or IPROTO_ROLLBACK with the current value of stream.stream_id; with *box.begin()* and *box.commit()* or *box.rollback()*; with SQL and *START TRANSACTION* and *COMMIT* or *ROLLBACK*. An application can use any or all of these ways.

### Events and subscriptions

The commands below support asynchronous server-client notifications signalled with *box.broadcast()*. Servers that support the new feature set the `IPROTO_FEATURE_WATCHERS` feature in reply to the `IPROTO_ID` command. When the connection is closed, all watchers registered for it are unregistered.

The remote watcher (event subscription) protocol works in the following way:

1. The client sends an `IPROTO_WATCH` packet to subscribe to the updates of a specified key defined on the server.
2. The server sends an `IPROTO_EVENT` packet to the subscribed client after registration. The packet contains the key name and its current value. After that, the packet is sent every time the key value is updated with `box.broadcast()`, provided that the last notification was acknowledged (see below).
3. After receiving the notification, the client sends an `IPROTO_WATCH` packet to acknowledge the notification.
4. If the client doesn't want to receive any more notifications, it unsubscribes by sending an `IPROTO_UNWATCH` packet.

All the three request types are asynchronous – the receiving end doesn't send a packet in reply to any of them. Therefore, neither of them has a sync number.

### IPROTO_WATCH

Code: 0x4a.

Register a new watcher for the given notification key or confirms a notification if the watcher is already subscribed. The watcher is notified after registration. After that, the notification is sent every time the key is updated. The server doesn't reply to the request unless it fails to parse the packet.

### IPROTO_UNWATCH

Code: 0x4b.

Unregister a watcher subscribed to the given notification key. The server doesn't reply to the request unless it fails to parse the packet.

### IPROTO_EVENT

Code: 0x4c.

Sent by the server to notify a client about an update of a key.

`IPROTO_EVENT_DATA` contains data sent to a remote watcher. The parameter is optional, the default value is `MP_NIL`.

#### Graceful shutdown protocol

##### Overview

Since *2.10.0*.

The graceful shutdown protocol is a mechanism that helps to prevent data loss in requests in case of a shutdown command. According to the protocol, when a server receives an `os.exit()` command or a `SIGTERM` signal, it does not exit immediately. Instead of that, first, the server stops listening for new connections. Then, the server sends the shutdown packets to all connections that support the graceful shutdown protocol. When a client is notified about the upcoming server exit, it stops serving any new requests and waits for active requests to complete before closing the connections. Once all connections are terminated, the server will be shut down.

The protocol uses the event subscription system. That is, the feature is available if the server supports the *box.shutdown* event and `IPROTO_WATCH`. For more information about it, see *reference for the event watchers* and the *corresponding page in the Binary Protocol section*.

##### How the graceful shutdown works

The shutdown protocol works in the following way:

1. First, the server receives a shutdown request. It can be either an `os.exit()` command or a *SIGTERM* signal.
2. Then the *box.shutdown* event is generated. The server broadcasts it to all subscribed remote watchers (see *IPROTO_WATCH*). That is, the server calls *box.broadcast(„box.shutdown“, true)* from the *box.ctl.on_shutdown()* trigger callback. Once this is done, the server stops listening for new connections.
3. From now on, the server waits until all subscribed connections are terminated.
4. At the same time, the client gets the `box.shutdown` event and shuts the connection down gracefully.
5. After all connections are closed, the server will be stopped. Otherwise, a timeout occurs, and the Tarantool exits immediately. You can set up the required timeout with the *set_on_shutdown_timeout()* function.

### SQL-specific requests and responses

Below are considered the *IPROTO_EXECUTE* and *IPROTO_PREPARE* requests, followed by a description of *responses*.

### Basic request description

| Name | Code | Description |
|---|---|---|
| *IPROTO_EXECUTE* | 0x0b | Execute an SQL statement (*box.execute()*) |
| *IPROTO_PREPARE* | 0x0d | Prepare an SQL statement (*box.prepare()*) |

## IPROTO_EXECUTE

Code: 0x0b.

The body is a 3-item map:

- Use IPROTO_STMT_ID (0x43) and statement-id (MP_INT) if executing a prepared statement. Use IPROTO_SQL_TEXT (0x40) and statement-text (MP_STR) if executing an SQL string.
- IPROTO_SQL_BIND (0x41) corresponds to the array of parameter values to match ? placeholders or :name placeholders.
- IPROTO_OPTIONS (0x2b) corresponds to the array of options. It is usually empty.

### Example 1

Suppose we prepare a statement with two ? placeholders, and execute with two parameters, thus:

```
n = conn:prepare[[VALUES (?, ?);]]
conn:execute(n.stmt_id, {1,'a'})
```

Then the body will look like this:

The *Understanding binary protocol* tutorial shows actual byte codes of the IPROTO_EXECUTE message.

To call a prepared statement with named parameters from a connector pass the parameters within an array of maps. A client should wrap each element into a map, where the key holds a name of the parameter (with a colon) and the value holds an actual value. So, to bind foo and bar to 42 and 43, a client should send `IPROTO_SQL_TEXT: <...>, IPROTO_SQL_BIND: [{"foo": 42}, {"bar": 43}]`.

If a statement has both named and non-named parameters, wrap only named ones into a map. The rest of the parameters are positional and will be substituted in order.

#### Example 2

Let's ask for full metadata and then select the two rows from a table named t1 that has columns named DD and Д:

```
conn.space._session_settings:update('sql_full_metadata', {{'=', 'value', true}})
conn:prepare([[SELECT dd, дд AS д FROM t1;]])
```

In the iproto request, there would be no IPROTO_DATA and there would be two additional items:

- `34 00 = IPROTO_BIND_COUNT and MP_UINT = 0` (there are no parameters to bind).
- `33 90 = IPROTO_BIND_METADATA and MP_ARRAY, size 0` (there are no parameters to bind).

Here is what the request body looks like:

### IPROTO_PREPARE

Code: 0x0d.

The body is a 1-item map:

The IPROTO_PREPARE map item is the same as the first item of the *IPROTO_EXECUTE* body for an SQL string.

#### Responses for SQL

After the *header*, for a response to an SQL statement, there will be a body that is slightly different from the body for non-SQL requests/responses.

#### Responses to SELECT, VALUES, or PRAGMA

If the SQL statement is SELECT or VALUES or PRAGMA, the response contains:

#### Example

Let's ask for full metadata and then select the two rows from a table named t1 that has columns named DD and Д:

```
conn.space._session_settings:update('sql_full_metadata', {{'=', 'value', true}})
conn:execute([[SELECT dd, дд AS д FROM t1;]])
```

The response body might look like this:

The tutorial *Understanding the binary protocol* shows actual byte codes of responses to the above SQL messages.

### Responses to other requests

If the SQL request is not SELECT or VALUES or PRAGMA, then the response body contains only IPROTO_SQL_INFO (0x42). Usually IPROTO_SQL_INFO is a map with only one item – SQL_INFO_ROW_COUNT (0x00) – which is the number of changed rows.

For example, if the request is `INSERT INTO table-name VALUES (1), (2), (3)`, then the response body contains an `IPROTO_SQL_INFO` map with `SQL_INFO_ROW_COUNT = 3`.

The IPROTO_SQL_INFO map may contain a second item – `SQL_INFO_AUTO_INCREMENT_IDS (0x01)` – which is the new primary-key value (or values) for an INSERT in a table defined with PRIMARY KEY AUTOINCREMENT. In this case the MP_MAP will have two keys, and one of the two keys will be 0x01: SQL_INFO_AUTO_INCREMENT_IDS, which is an array of unsigned integers.

### Replication requests and responses

This section describes internal requests and responses that happen during replication. Each of them is distinguished by the header, containing a unique *IPROTO_REQUEST_TYPE* value. These values and the corresponding packet body structures are considered below.

Connectors and clients do not need to send replication packets.

### General

| Name | Code | Description |
|---|---|---|
| *IPROTO_JOIN* | 0x41 | Request to join a replica set |
| *IPROTO_SUBSCRIBE* | 0x42 | Request to subscribe to a specific node in a replica set |
| *IPROTO_VOTE* | 0x44 | Request for replication |
| *IPROTO_BALLOT* | 0x29 | Response to IPROTO_VOTE. Used during replica set bootstrap |
| IPROTO_FETCH_SNAPSHOT | 0x45 | Fetch the master's snapshot and start anonymous replication. See *replication_anon* |
| IPROTO_REGISTER | 0x46 | Register an anonymous replica so it is not anonymous anymore |

The master also sends *heartbeat* messages to the replicas. The heartbeat message's IPROTO_REQUEST_TYPE is 0.

Below are details on individual replication requests. For synchronous replication requests, see *below*.

### Heartbeats

Once in *replication_timeout* seconds, a master sends a *heartbeat* message to a replica, and the replica sends a response. Both messages" IPROTO_REQUEST_TYPE is IPROTO_OK. IPROTO_TIMESTAMP is a float-64 MP_DOUBLE 8-byte timestamp.

Since version 2.11, both messages have an optional field in the body that contains the *IPROTO_VCLOCK_SYNC* key. The master's heartbeat has no body if the IPROTO_VCLOCK_SYNC key is omitted.

The message from master to a replica:

The response from the replica:

The tutorial *Understanding the binary protocol* shows actual byte codes of the above heartbeat examples.

### IPROTO_JOIN

Code: 0x41.

To join a replica set, an instance must send an initial IPROTO_JOIN request to any node in the replica set:

The node that receives the request does the following in response:

1. It sends its vclock:
2. It sends a number of *INSERT* requests (with additional LSN and ServerID). In this way, the data is updated on the instance that sent the IPROTO_JOIN request. The instance should not reply to these INSERT requests.
3. It sends the new vclock's MP_MAP in a response similar to the one above and closes the socket.

### IPROTO_SUBSCRIBE

Code: 0x42.

If *IPROTO_JOIN* was successful, the initiator instance must send an IPROTO_SUBSCRIBE request to all the nodes listed in its *box.cfg.replication*:

After a successful IPROTO_SUBSCRIBE request, the instance must process every request that could come from other masters. Each master's request includes a vclock pair corresponding to that master – its instance ID and its LSN, independent from other masters.

IPROTO_ID_FILTER (0x51) is an optional key used in the SUBSCRIBE request followed by an array of ids of instances whose rows won't be relayed to the replica. The field is encoded only when the ID list is not empty.

### IPROTO_VOTE

Code: 0x44.

When connecting for replication, an instance sends an IPROTO_VOTE request. It has no body:

IPROTO_VOTE is critical during replica set bootstrap. The response to this request is *IPROTO_BALLOT*.

### IPROTO_BALLOT

Code: 0x29.

This value of IPROTO_REQUEST_TYPE indicates a message sent in response to *IPROTO_VOTE* (not to be confused with the key IPROTO_RAFT_VOTE).

IPROTO_BALLOT and IPROTO_VOTE are critical during replica set bootstrap. IPROTO_BALLOT corresponds to a map containing the following fields:

IPROTO_BALLOT_REGISTERED_REPLICA_UUIDS has the MP_ARRAY type. The array contains MP_STR elements.

### Synchronous

| Name | Code | Description |
|---|---|---|
| *IPROTO_RAFT* | 0x1e | Inform that the node changed its *RAFT status* |
| *IPROTO_RAFT_PROMOTE* | 0x1f | Wait, then choose new replication leader |
| *IPROTO_RAFT_DEMOTE* | 0x20 | Revoke the leader role from the instance |
| *IPROTO_RAFT_CONFIRM* | 0x28 | Confirm that the RAFT transactions have achieved quorum and can be committed |
| *IPROTO_RAFT_ROLLBACK* | 0x29 | Roll back the RAFT transactions because they haven't achieved quorum |

#### IPROTO_RAFT

Code: 0x1e.

A node broadcasts the IPROTO_RAFT request to all the replicas connected to it when the RAFT state of the node changes. It can be any actions changing the state, like starting a new election, bumping the term, voting for another node, becoming the leader, and so on.

If there should be a response, for example, in case of a vote request to other nodes, the response will also be an IPROTO_RAFT message. In this case, the node should be connected as a replica to another node from which the response is expected because the response is sent via the replication channel. In other words, there should be a full-mesh connection between the nodes.

IPROTO_REPLICA_ID is the ID of the replica from which the request came.

#### IPROTO_RAFT_PROMOTE

Code: 0x1f.

See *box.ctl.promote()*.

In the header:

- IPROTO_REPLICA_ID is the replica ID of the node that sent the request.
- IPROTO_LSN is the actual LSN of the promote operation as recorded in the WAL.

In the body:

- IPROTO_REPLICA_ID is the replica ID of the previous *synchronous queue owner*.
- IPROTO_LSN is the LSN of the last operation on the previous synchronous queue owner.
- IPROTO_TERM is the term in which the node that sent the request becomes the synchronous queue owner. This term corresponds to the value of *box.info.synchro.queue.term* on the instance.

### IPROTO_RAFT_DEMOTE

Code: 0x20.

See *box.ctl.demote()*.

In the header:

- IPROTO_REPLICA_ID is the replica ID of the node that sent the request.
- IPROTO_LSN is the actual LSN of the demote operation as recorded in the WAL.

In the body:

- IPROTO_REPLICA_ID is the replica ID of the node that sent the request (same as the value in the header).
- IPROTO_LSN is the LSN of the last synchronous transaction recorded in the node's WAL.
- IPROTO_TERM is the term in which the queue becomes empty.

### IPROTO_RAFT_CONFIRM

Code: 0x28.

This message is used in replication connections between Tarantool nodes in *synchronous replication*. It is not supposed to be used by any client applications in their regular connections.

This message confirms that the transactions that originated from the instance with id = IPROTO_REPLICA_ID (body) have achieved quorum and can be committed, up to and including LSN = IPROTO_LSN (body).

The body is a 2-item map:

In the header:

- IPROTO_REPLICA_ID is the ID of the replica that sends the confirm message.
- IPROTO_LSN is the LSN of the confirmation action.

In the body:

- IPROTO_REPLICA_ID is the ID of the instance from which the transactions originated.
- IPROTO_LSN is the LSN up to which the transactions should be confirmed.

Prior to Tarantool v. 2.10.0, IPROTO_RAFT_CONFIRM was called IPROTO_CONFIRM.

### IPROTO_RAFT_ROLLBACK

Code: 0x29.

This message is used in replication connections between Tarantool nodes in *synchronous replication*. It is not supposed to be used by any client applications in their regular connections.

This message says that the transactions that originated from the instance with id = IPROTO_REPLICA_ID (body) couldn't achieve quorum for some reason and should be rolled back, down to LSN = IPROTO_LSN (body) and including it.

The body is a 2-item map:

In the header:

- IPROTO_REPLICA_ID is the ID of the replica that sends the rollback message.
- IPROTO_LSN is the LSN of the rollback action.

In the body:

- IPROTO_REPLICA_ID is the ID of the instance from which the transactions originated.
- IPROTO_LSN is the LSN starting with which all pending synchronous transactions should be rolled back.

Prior to Tarantool v. 2.10.0, IPROTO_RAFT_ROLLBACK was called IPROTO_ROLLBACK.

### 9.8.2 Дополнительные типы MessagePack

Tarantool использует предопределенные дополнительные типы MessagePack для представления некоторых специальных значений. Дополнительные типы включают `MP_DECIMAL`, `MP_UUID`,``MP_ERROR``, `MP_DATETIME`, and `MP_INTERVAL`. Эти типы требуют особого внимания со стороны разработчиков коннекторов, так как должны рассматриваться отдельно от типов MessagePack по умолчанию и корректно приводиться к типам языков программирования.

#### Тип DECIMAL

Тип MessagePack EXT `MP_EXT` вместе с типом расширения `MP_DECIMAL` является заголовком для значений типа DECIMAL.

MP_DECIMAL – это 1.

Спецификация MessagePack определяет два вида типов:

- типы `fixext 1/2/4/8/16` имеют фиксированную длину, поэтому длина не кодируется явно;
- типы `ext 8/16/32` требуют кодирования длины данных.

`MP_EXP` + (не обязательно) `length` подразумевает использование одного из этих типов.

Десятичное представление MessagePack выглядит следующим образом:

```
+--------+------------------+------------+================+
| MP_EXT | length (optional) | MP_DECIMAL | PackedDecimal |
+--------+------------------+------------+================+
```

Здесь `length` – это длина поля `PackedDecimal`, и оно имеет тип `MP_UINT`, когда кодируется явно (т.е. когда тип – `ext 8/16/32`).

`PackedDecimal` имеет следующую структуру:

```
 <--- длина в байтах -->
+-------+==============+
| scale |     BCD      |
+-------+==============+
```

Здесь `scale` – это либо `MP_INT`, либо `MP_UINT`. `scale` = количество цифр после запятой.

`BCD` - это последовательность байтов, обозначающая десятичные цифры кодируемого числа (каждый байт имеет две десятичные цифры, каждая из которых кодируется с использованием 4-битных `nibbles`), поэтому `byte >> 4` - первая цифра, а `byte & 0x0f` - вторая цифра. Самая левая цифра в массиве - самая значимая. Самая правая цифра в массиве - наименее значимая.

Первый байт `BCD`-массива содержит первую цифру числа, представленную следующим образом:

```
|  4 bits           |  4 bits           |
   = 0x               = the 1st digit
```

(Первый `nibble` содержит 0, если десятичное число имеет четное число цифр). Последний байт BCD-массива содержит последнюю цифру числа и последний `nibble`, представленный следующим образом:

```
|  4 bits           |  4 bits           |
   = the last digit    = nibble
```

Последний `nibble` обозначает знак числа:

- `0x0a`, `0x0c`, `0x0e`, `0x0f` обозначают плюс,
- `0x0b` and `0x0d` обозначают минус.

**Примеры**

Десятичное число `-12.34` будет закодировано как `0xd6,0x01,0x02,0x01,0x23,0x4d`:

```
|MP_EXT (fixext 4) | MP_DECIMAL | scale |  1    |  2,3 |  4 (minus) |
|       0xd6       |    0x01    | 0x02  | 0x01  | 0x23 | 0x4d       |
```

Десятичное число 0.0000000000000000000000000000000010 будет закодировано как `0xc7,0x03,0x01,0x24,0x01,0x0c`:

```
| MP_EXT (ext 8) | length | MP_DECIMAL | scale |  1    | 0 (plus) |
|      0xc7      |  0x03  |    0x01    | 0x24  | 0x01  | 0x0c     |
```

### Тип UUID

Тип MessagePack EXT `MP_EXT` вместе с типом расширения `MP_UUID` является заголовком для значений типа UUID. Доступно с версии *2.4.1*.

MP_UUID – это 2.

Спецификация MessagePack определяет `d8` как `fixext` с размером 16, а размер `UUID` всегда равен 16. Таким образом, представление `UUID` в MessagePack выглядит следующим образом:

```
+--------+------------+-----------------+
| MP_EXT | MP_UUID    | UuidValue       |
| = d8   | = 2        | = 16-byte value |
+--------+------------+-----------------+
```

The 16-byte value has 2 digits per byte. Typically, it consists of 11 fields, which are encoded as big-endian unsigned integers in the following order:

- `time_low` (4 bytes)
- `time_mid` (2 bytes)
- `time_hi_and_version` (2 bytes)
- `clock_seq_hi_and_reserved` (1 byte)
- `clock_seq_low` (1 byte)
- `node[0]`, . . . , `node[5]` (1 byte each)

Некоторые функции в *модуле UUID* могут выдавать значения, совместимые с типом данных UUID. Например, после

```
uuid = require('uuid')
box.schema.space.create('t')
box.space.t:create_index('i', {parts={1,'uuid'}})
box.space.t:insert{uuid.fromstr('f6423bdf-b49e-4913-b361-0740c9702e4b')}
box.space.t:select()
```

пакет ответа сервера покажет, что он содержит

```
d8 02 f6 42 3b df b4 9e 49 13 b3 61 07 40 c9 70 2e 4b
```

### Тип ERROR

Начиная с версии *2.4.1*, в ответах на ошибки содержится дополнительная информация, соответствующая описанию в разделе *Бинарный протокол — ответы на ошибки*. Это «совместимое» улучшение, потому что клиенты, которые ожидают ответы сервера старого образца, должны игнорировать компоненты ассоциативного массива, которые они не распознают. Обратите внимание, что константа `IPROTO_ERROR` в `./box/iproto_constants.h` была `0x31`, а теперь `IPROTO_ERROR` – `0x52`, а `IPROTO_ERROR_24` – `0x31`.

MP_ERROR – это 3.

```
++=========================+============================+
||                         |                            |
||   0x31: IPROTO_ERROR_24 |   0x52: IPROTO_ERROR                                    |
|| MP_INT: MP_STRING       | MP_MAP: дополнительная информация  |
||                         |                            |
++=========================+============================+
                         MP_MAP
```

Дополнительная информация, большая часть которой также хранится в полях *объекта ошибки*:

`MP_ERROR_TYPE` (0x00) (MP_STR) Тип, подразумевающий источник, как в *error_object*.base_type, например «ClientError».

`MP_ERROR_FILE` (0x01) (MP_STR) Файл с исходным кодом, в котором была перехвачена ошибка, как в *error_object*.trace.

`MP_ERROR_LINE` (0x02) (MP_UINT) Номер строки в файле исходных кодов, как в *error_object*.trace.

`MP_ERROR_MESSAGE` (0x03) (MP_STR) Текст причины, как в *error_object*.message. Значение здесь будет таким же, как и значение `IPROTO_ERROR_24`

`MP_ERROR_ERRNO` (0x04) (MP_UINT) Порядковый номер ошибки, как в *error_object*.errno. Не путать с `MP_ERROR_ERRCODE`.

`MP_ERROR_ERRCODE` (0x05) (MP_UINT) Номер ошибки, как в файле errcode.h, как в *error_object*.code, который также можно получить функцией C *box_error_code()*. Значение здесь будет таким же, как и в нижней части значения Response-Code-Indicator.

`MP_ERROR_FIELDS` (0x06) (MP_MAPs) Дополнительные поля в зависимости от типа ошибки. Например, если `MP_ERROR_TYPE` имеет значение «AccessDeniedError», то `MP_ERROR_FIELDS` будет включать «object_type», «object_name», «access_type». При отсутствии дополнительных полей это поле будет пропущено в теле ответа.

Разработчики клиента и коннекторов должны убедиться, что неизвестные ключи ассоциативных массивов игнорируются, а также проверить наличие новых ключей в файле исходного кода Tarantool,

в котором определено создание объекта ошибки. В версии 2.4.1 имя этого файла с исходным кодом mp_error.cc.

Например, в версии 2.4.1 или более поздней, если мы попытаемся создать дубликат пробела с помощью команды `conn:eval([[box.schema.space.create('_space');]])`, ответ сервера будет выглядеть так:

```
ce 00 00 00 88                  MP_UINT = HEADER + BODY SIZE
83                              MP_MAP, size 3 (i.e. 3 items in header)
  00                              Response-Code-Indicator
  ce 00 00 80 0a                  MP_UINT = hexadecimal 800a
  01                              IPROTO_SYNC
  cf 00 00 00 00 00 00 00 05      MP_UINT = sync value
  05                              IPROTO_SCHEMA_VERSION
  ce 00 00 00 4e                  MP_UINT = schema version value
82                              MP_MAP, size 2
  31                              IPROTO_ERROR_24
  bd 53 70 61 63 etc.             MP_STR = "Space '_space' already exists"
  52                              IPROTO_ERROR
  81                              MP_MAP, size 1
    00                              MP_ERROR_STACK
    91                              MP_ARRAY, size 1
      86                              MP_MAP, size 6
        00                              MP_ERROR_TYPE
        ab 43 6c 69 65 6e 74 etc.       MP_STR = "ClientError"
        02                              MP_ERROR_LINE
        cd                              MP_UINT = line number
        01                              MP_ERROR_FILE
        aa 01 b6 62 75 69 6c etc.       MP_STR "builtin/box/schema.lua"
        03                              MP_ERROR_MESSAGE
        bd 53 70 61 63 65 20 etc.       MP_STR = Space.'_space'.already.exists"
        04                              MP_ERROR_ERRNO
        00                              MP_UINT = error number
        05                              MP_ERROR_ERRCODE
        0a                              MP_UINT = eror code ER_SPACE_EXISTS
```

### The DATETIME type

Since version *2.10.0*. The MessagePack EXT type `MP_EXT` together with the extension type `MP_DATETIME` is a header for values of the DATETIME type. It creates a container with a payload of 8 or 16 bytes.

`MP_DATETIME` type is 4.

The MessagePack specification defines `d7` to mean `fixext` with size 8 or `d8` to mean `fixext` with size 16.

So the datetime MessagePack representation looks like this:

```
+---------+----------------+==========+-----------------+
| MP_EXT  | MP_DATETIME    | seconds  | nsec; tzoffset; |
| = d7/d8 | = 4            |          | tzindex;        |
+---------+----------------+==========+-----------------+
```

MessagePack data contains:

- Seconds (8 bytes) as an unencoded 64-bit signed integer stored in the little-endian order.

- The optional fields (8 bytes), if any of them have a non-zero value. The fields include `nsec`, `tzoffset`, and `tzindex` packed in the little-endian order.

For more information about the datetime type, see *datetime field type details* and *reference for the datetime module*.

### The INTERVAL type

Since version *2.10.0*. The MessagePack EXT type `MP_EXT` together with the extension type `MP_INTERVAL` is a header for values of the INTERVAL type.

`MP_INTERVAL` type is 6.

The interval is saved as a variant of a map with a predefined number of known attribute names. If some attributes are undefined, they are omitted from the generated payload.

The interval MessagePack representation looks like this:

```
+--------+-------------------------+-------------+----------------+
| MP_EXT | Size of packed interval | MP_INTERVAL | PackedInterval |
+--------+-------------------------+-------------+----------------+
```

Packed interval consists of:

- Packed number of non-zero fields.
- Packed non-null fields.

Each packed field has the following structure:

```
+----------+======================+
| field ID |     field value      |
+----------+======================+
```

The number of defined (non-null) fields can be zero. In this case, the packed interval will be encoded as integer 0.

List of the field IDs:

- 0 – year
- 1 – month
- 2 – week
- 3 – day
- 4 – hour
- 5 – minute
- 6 – second
- 7 – nanosecond
- 8 – adjust

**Example**

Interval value `1 years, 200 months, -77 days` is encoded in the following way:

```
tarantool> I = datetime.interval.new{year = 1, month = 200, day = -77}
---
...

tarantool> I
---
- +1 years, 200 months, -77 days
...

tarantool> M = msgpack.encode(I)
---
...

tarantool> M
---
- !!binary xwsGBAABAczIA9CzCAE=
...

tarantool> tohex = function(s) return (s:gsub('.', function(c) return string.format('
↪%02X ', string.byte(c)) end)) end
---
...

tarantool> tohex(M)
---
- 'C7 0B 06 04 00 01 01 CC C8 03 D0 B3 08 01 '
...
```

Where:

- C7 – MP_EXT
- 0B – size of a packed interval value (11 bytes)
- 06 – MP_INTERVAL type
- 04 – number of defined fields
- 00 – field ID (year)
- 01 – packed value 1
- 01 – field ID (month)
- CCC8 – packed value 200
- 03 – field ID (day)
- D0B3 – packed value -77
- 08 – field ID (adjust)
- 01 – packed value 1 (DT_LIMIT)

For more information about the interval type, see *interval field type details* and *description of the datetime module*.

### 9.8.3 Форматы файлов

#### The WAL file format

To maintain *data persistence*, Tarantool writes each data change request (insert, update, delete, replace, upsert) to a write-ahead log (WAL) file in the *wal.dir* directory. Each data change request is assigned a continuously growing 64-bit log sequence number. The name of the WAL file is based on the log sequence number of the first record in the file, plus an extension `.xlog`. A new WAL file is created when the current one reaches the *wal_max_size* size.

Each WAL record contains:

- a log sequence number
- a data change request (formatted as in *Tarantool's binary protocol*)
- a header
- some metadata
- the data formatted according to msgpack rules.

To see the hexadecimal bytes of the given WAL file, use the `hexdump` command:

```
$ hexdump 00000000000000000000.xlog
```

For example, the WAL file after the first INSERT request might look the following way:

```
Шестнадцатеричный дамп WAL-файла       Комментарий
--------------------       -------
58 4c 4f 47 0a             "XLOG\n"
30 2e 31 33 0a             "0.13\n" = version
53 65 72 76 65 72 3a 20    "Server: "
38 62 66 32 32 33 65 30 2d [Server UUID]\n
36 39 31 34 2d 34 62 35 35
2d 39 34 64 32 2d 64 32 62
36 64 30 39 62 30 31 39 36
0a
56 43 6c 6f 63 6b 3a 20    "Vclock: "
7b 7d                      "{}" = vclock value, initially blank
...                        (not shown = tuples for system spaces)
d5 ba 0b ab                Magic row marker always = 0xab0bbad5
19                         Length, not including length of header, = 25 bytes
00                           Record header: previous crc32
ce 8c 3e d6 70               Record header: current crc32
a7 cc 73 7f 00 00 66 39      Record header: padding
84                         msgpack code meaning "Map of 4 elements" follows
00 02                         element#1: tag=request type, value=0x02=IPROTO_INSERT
02 01                         element#2: tag=server id, value=0x01
03 04                         element#3: tag=lsn, value=0x04
04 cb 41 d4 e2 2f 62 fd d5 d4 element#4: tag=timestamp, value=an 8-byte "Float64"
82                         msgpack code meaning "map of 2 elements" follows
10 cd 02 00                   element#1: tag=space id, value=512, big byte first
21 91 01                      element#2: tag=tuple, value=1-element fixed array={1}
```

Tarantool processes requests atomically: a change is either accepted and recorded in the WAL, or discarded completely. To clarify how this happens, see the example with the REPLACE request below:

1. Экземпляр сервера пытается найти оригинальный кортеж по первичному ключу. Если кортеж найден, ссылка на него сохраняется для дальнейшего использования.
2. Происходит проверка нового кортежа. Например, если в нем нет проиндексированного поля, или же тип проиндексированного поля не совпадает с типом в определении индекса, изменение прерывается.
3. Новый кортеж заменяет старый кортеж во всех существующих индексах.
4. В процесс записи WAL, запущенный в отдельном потоке, отправляется сообщение о необходимости внесения записи в WAL-файл. Экземпляр переключается на работу со следующим запросом, пока запись не будет подтверждена.
5. On success, a confirmation is sent to the client. On failure, a rollback procedure begins. During the rollback procedure, the transaction processor rolls back all changes to the database which occurred after the first failed change, from latest to oldest, up to the first failed change. All rolled back requests are aborted with `ER_WAL_IO` error. No new change is applied while rollback is in progress. When the rollback procedure is finished, the server restarts the processing pipeline.

Одно из преимуществ описанного алгоритма заключается в том, что достигается полная обработка запроса по конвейеру даже для запросов с одинаковым значением первичного ключа. В результате производительность базы данных не падает, даже если все запросы относятся к одному ключу в одном спейсе.

The transaction processor thread communicates with the WAL writer thread using asynchronous (yet reliable) messaging. The transaction processor thread, not being blocked on WAL tasks, continues to handle requests quickly even at high volumes of disk I/O. A response to a request is sent as soon as it is ready, even if there were earlier incomplete requests on the same connection. In particular, SELECT performance, even for SELECTs running on a connection packed with UPDATEs and DELETEs, remains unaffected by disk load.

The WAL writer employs a number of durability modes, as defined in configuration variable *wal.mode*. It is possible to turn the write-ahead log completely off, by setting the `wal_mode` option to *none*. Even without the write-ahead log it's still possible to take a persistent copy of the entire data set with the *box.snapshot()* request.

An `.xlog` file always contains changes based on the primary key. Even if the client requested an update or delete using a secondary key, the record in the .xlog file contains the primary key.

### Формат файла снимка

The format of a snapshot (`.snap`) file is the following:

- The snapshot header contains the instance's global unique identifier and the snapshot file's position in history, relative to earlier snapshot files.
- The snapshot content contains the records of inserts to memtx spaces. That differs from the content of an `.xlog` file that may contain records for any data-change requests (inserts, updates, upserts, and deletes).

Primarily, the records in the snapshot file have the following order:

- System spaces (id >= 256 && id <= 511), ordered by ID.
- Non-system spaces, ordered by ID.

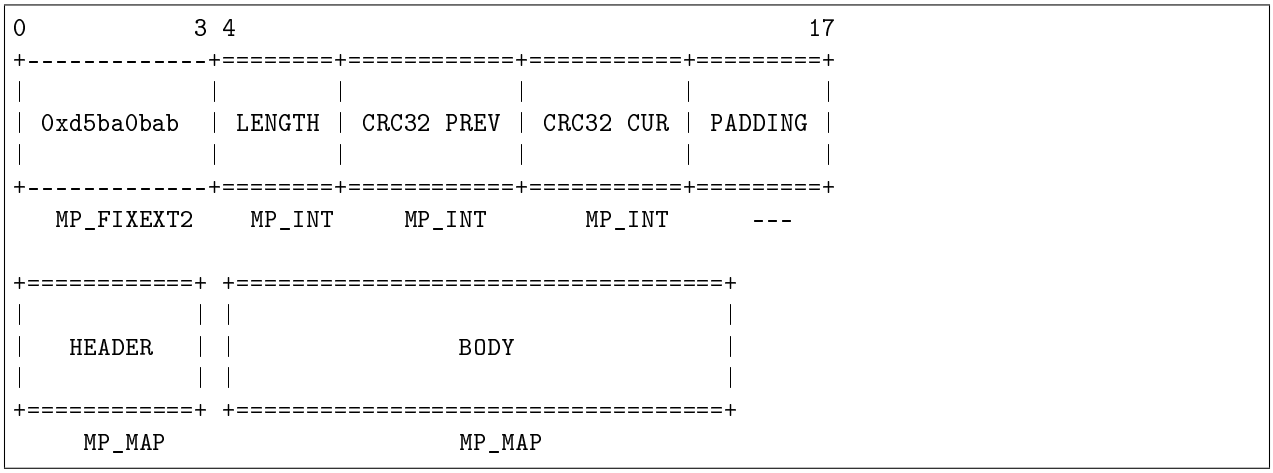
Secondarily, the .snap file's records are ordered by primary key within space ID.

#### Example

The header of a `.snap` or `.xlog` file might look in the following way:

```
<type>\n                  SNAP\n или XLOG\n
<version>\n               в данный момент 0.13\n
Server: <server_uuid>\n   где UUID -- это 36-байтная строка
VClock: <vclock_map>\n    например, {1: 0}\n
\n
```

После файла заголовка идут кортежи с данными. Кортежи начинаются с маркера строки `0xd5ba0bab`, а после последнего кортежа может стоять маркер конца файла `0xd510aded`. Таким образом, между заголовком файла и маркером конца файла могут быть кортежи с данными в следующем виде:




### 9.8.4 Процесс восстановления

Процесс восстановления начинается, когда `box.cfg{}` впервые используется после запуска экземпляра Tarantool-сервера.

Процесс восстановления должен восстановить базы данных на момент последнего отключения экземпляра. Для этого можно использовать последний файл снимка и любые WAL-файлы, которые были записаны после создания снимка. Ситуацию осложняет фактор того, что в Tarantool используются два движка – данные memtx'а должны быть реконструированы полностью из снимка и WAL-файлов, тогда как данные vinyl'а будут находиться на диске, но может потребоваться их обновление на время создания контрольной точки. (При создании снимка Tarantool передает движку vinyl команду создания контрольной точки, а операция создания снимка откатывается в случае какой-либо ошибки, поэтому контрольная точка vinyl'а будет настолько же актуальной, как и файл снимка.)

**Шаг 1**

Выполнить чтение конфигурационных параметров из запроса `box.cfg{}`. Параметры, которые могут повлиять на восстановление: *work_dir*, *wal_dir*, *memtx_dir*, *vinyl_dir* и *force_recovery*.

**Шаг 2**

Найти последний файл снимка. Использовать данные для реконструкции in-memory баз данных. Передать команду vinyl'у о восстановлении до последней контрольной точки.

На самом деле, есть два варианта реконструкции баз данных memtx'а в зависимости от того, выполняется ли стандартная процедура.

Если выполняется стандартная процедура (`force_recovery = false`), memtx может выполнить чтение данных из снимка с отключенными индексами. Сначала все кортежи считываются в память. Затем происходит массовая загрузка первичных ключей с учетом того, что данные уже отсортированы по первичному ключу в каждом спейсе.

Если выполняется нестандартная процедура принудительного восстановления (`force_recovery = true`), Tarantool проводит дополнительную проверку. Сначала индексы активны, и кортежи добавляются по одному. Это означает, что будут выявлены любые нарушения ограничений уникальности ключей, и все повторяющиеся значения пропускаются. Как правило, не будет нарушений ограничений или повторяющихся значений, поэтому такие проверки проводятся только в случае ошибки.

**Шаг 3**

Найти WAL-файл, который был создан во время создания файла снимка или позже. Выполнить чтение записей журнала до тех пор, пока LSN записи в журнале не будет больше LSN снимка или больше LSN контрольной точки в vinyl'е. Это и будет начальной точкой для процесса восстановления, которая соответствует текущему состоянию движков.

**Шаг 4**

Повторить записи журнала с начальной точки до конца WAL. Движок пропускает команду повторения, если данные старше контрольной точки движка.

**Шаг 5**

Повторно создать все вторичные индексы для движка memtx.

### 9.8.5 Replication internals

#### Server startup with replication

In addition to the recovery process described in the section *Recovery process*, the server must take additional steps and precautions if *replication* is enabled.

Once again the startup procedure is initiated by the `box.cfg{}` request. One of the `box.cfg` parameters may be *replication* which specifies replication source(-s). We will refer to this replica, which is starting up due to `box.cfg`, as the «local» replica to distinguish it from the other replicas in a replica set, which we will refer to as «distant» replicas.

*If there is no snapshot .snap file and the* `replication` *parameter is empty and* `cfg.read_only=false`: then the local replica assumes it is an unreplicated «standalone» instance, or is the first replica of a new replica set. It will generate new UUIDs for itself and for the replica set. The replica UUID is stored in the `_cluster` space; the replica set UUID is stored in the `_schema` space. Since a snapshot contains all the data in all the spaces, that means the local replica's snapshot will contain the replica UUID and the replica set UUID. Therefore, when the local replica restarts on later occasions, it will be able to recover these UUIDs when it reads the .snap file.

*If there is no snapshot .snap file and the* `replication` *parameter is empty and* `cfg.read_only=true`: it cannot be the first replica of a new replica set because the first replica must be a master. Therefore an error message will occur: ER_BOOTSTRAP_READONLY. To avoid this, change the setting for this (local) instance to `read_only = false`, or ensure that another (distant) instance starts first and has the local instance's UUID in its `_cluster` space. In the latter case, if ER_BOOTSTRAP_READONLY still occurs, set the local instance's *box.replication_connect_timeout* to a larger value.

*If there is no snapshot .snap file and the* `replication` *parameter is not empty and the* `_cluster` *space contains no other replica UUIDs*: then the local replica assumes it is not a standalone instance, but is not yet part of a replica set. It must now join the replica set. It will send its replica UUID to the first distant

replica which is listed in `replication` and which will act as a master. This is called the «join request». When a distant replica receives a join request, it will send back:

- the distant replica's replica set UUID,
- the contents of the distant replica's .snap file. When the local replica receives this information, it puts the replica set UUID in its `_schema` space, puts the distant replica's UUID and connection information in its `_cluster` space, and makes a snapshot containing all the data sent by the distant replica. Then, if the local replica has data in its WAL .xlog files, it sends that data to the distant replica. The distant replica will receive this and update its own copy of the data, and add the local replica's UUID to its `_cluster` space.

*If there is no snapshot .snap file and the* `replication` *parameter is not empty and the* `_cluster` *space contains other replica UUIDs*: then the local replica assumes it is not a standalone instance, and is already part of a replica set. It will send its replica UUID and replica set UUID to all the distant replicas which are listed in `replication`. This is called the «on-connect handshake». When a distant replica receives an on-connect handshake:

- the distant replica compares its own copy of the replica set UUID to the one in the on-connect handshake. If there is no match, then the handshake fails and the local replica will display an error.
- the distant replica looks for a record of the connecting instance in its `_cluster` space. If there is none, then the handshake fails. Otherwise the handshake is successful. The distant replica will read any new information from its own .snap and .xlog files, and send the new requests to the local replica.

In the end, the local replica knows what replica set it belongs to, the distant replica knows that the local replica is a member of the replica set, and both replicas have the same database contents.

*If there is a snapshot file and replication source is not empty*: first the local replica goes through the recovery process described in the previous section, using its own .snap and .xlog files. Then it sends a «subscribe» request to all the other replicas of the replica set. The subscribe request contains the server vector clock. The vector clock has a collection of pairs „server id, lsn“ for every replica in the `_cluster` system space. Each distant replica, upon receiving a subscribe request, will read its .xlog files“ requests and send them to the local replica if (lsn of .xlog file request) is greater than (lsn of the vector clock in the subscribe request). After all the other replicas of the replica set have responded to the local replica's subscribe request, the replica startup is complete.

The following temporary limitations applied for Tarantool versions earlier than 1.7.7:

- The URIs in the `replication` parameter should all be in the same order on all replicas. This is not mandatory but is an aid to consistency.
- The replicas of a replica set should be started up at slightly different times. This is not mandatory but prevents a situation where each replica is waiting for the other replica to be ready.

The following limitation still applies for the current Tarantool version:

- The maximum number of entries in the `_cluster` space is *32*. Tuples for out-of-date replicas are not automatically re-used, so if this 32-replica limit is reached, users may have to reorganize the `_cluster` space manually.

### Статус orphan (одиночный)

Starting with Tarantool version 1.9, there is a change to the procedure when an instance joins a replica set. During `box.cfg()` the instance tries to join all nodes listed in *box.cfg.replication*. If the instance does not succeed with connecting to the required number of nodes (see *bootstrap_strategy*), it switches to the **orphan** status. While an instance is in orphan status, it is read-only.

Чтобы «подключиться» к мастеру, реплика должна «установить соединение» с узлом мастера, а затем «выполнить синхронизацию».

«Установка соединения» означает контакт с мастером по физической сети и получение подтверждения. Если нет подтверждения соединения через *box.replication_connect_timeout* секунд (обычно 4 секунды), и повторные попытки подключения не сработали, то соединение не установлено.

«Синхронизация» означает получение обновлений от мастера для создания локальной копии базы данных. Синхронизация завершена, когда реплика получила все обновления или хотя бы получила достаточное количество обновлений, чтобы отставание реплики (см. *replication.upstream.lag* в `box.info()`) было меньше или равно количеству секунд, указанному в *box.cfg.replication_sync_lag*. Если значение `replication_sync_lag` не задано (nil) или указано как «TIMEOUT_INFINITY», то реплика пропускает шаг «синхронизация» и сразу же переходит на «отслеживание».

In order to leave orphan mode, you need to sync with a sufficient number of instances (*bootstrap_strategy*). To do so, you may either:

- Убрать из списка `box.cfg.replication` недоступные и прочие узлы, с которыми нельзя синхронизироваться.
- Вообще задать `""` (пустую строку) в качестве значения `box.cfg.replication`.

Возможны следующие ситуации.

**Ситуация 1: настройка**

Здесь впервые происходит вызов `box.cfg{}`. Реплика подключается, но набора реплик пока нет.

1. Set the status to „orphan“.
2. Try to connect to all nodes from `box.cfg.replication`. The replica tries to connect for the *replication_connect_timeout* number of seconds and retries each *replication_timeout* seconds if needed.
3. Abort and throw an error if a replica is not connected to the majority of nodes in `box.cfg.replication`.
4. Экземпляр может быть выбран в качестве лидера „leader“ в наборе реплик. Критерии выбора лидера включают в себя значение vclock (чем больше, тем лучше), а также доступность только для чтения или для чтения и записи (лучше всего для чтения и записи, кроме случаев, когда других вариантов нет). Лидер является мастером, к которому должны подключиться другие экземпляры. Лидер является мастером, который выполняет функции *box.once()*.
5. Если данный экземпляр выбран лидером набора реплик, выполняется «самонастройка»:

   a. Установка статуса „running“ (запущен).

   b. Возврат из `box.cfg{}`.

   В противном случае, данный экземпляр будет репликой, которая подключается к существующему набору реплик, поэтому:

   a. Настройка от лидера. См. примеры в разделе *Настройка набора реплик*.

   b. Синхронизация со всеми остальными узлами в наборе реплик в фоновом режиме.

**Ситуация 2: восстановление**

Здесь вызов `box.cfg{}` происходит не впервые, а повторно для осуществления восстановления.

1. Проведение *восстановления* из последнего локального снимка и WAL-файлов.
2. Try to establish connections to all other nodes for the *replication_connect_timeout* number of seconds. Once `replication_connect_timeout` is expired or all the connections are established, proceed to the «sync» state with all the established connections.
3. If connected, sync with all connected nodes, until the difference is not more than *replication_sync_lag* seconds.

**Ситуация 3: обновление конфигурации**

Здесь вызов `box.cfg{}` происходит не впервые, а повторно, поскольку изменились некоторые параметры репликации или что-то в наборе реплик.

1. Try to connect to all nodes from `box.cfg.replication`, within the time period specified in *replication_connect_timeout*.
2. Попытка синхронизации со всеми подключенными узлами в течение периода времени, указанного в *replication_sync_timeout*.
3. Если предыдущие шаги не выполнены, статус изменяется на „orphan“ (одиночный). (Попытки синхронизации будут продолжаться в фоновом режиме, и когда/если они будут успешны, статус „orphan“ отключится.)
4. Если предыдущие шаги выполнены, статус изменяется на „running“ (мастер) или „follow“ (реплика).

**Ситуация 4: повторная настройка**

Здесь не происходит вызов `box.cfg{}`. В определенный момент в прошлом реплика успешно установила соединение и в настоящий момент ожидает обновления от мастера. Однако мастер не может передать обновления, что может произойти случайно, или же если реплика работает слишком медленно (большое значение *lag*), а WAL-файлы (.xlog) с обновлениями были удалены. Такая ситуация не является критической – реплика может сбросить ранее полученные данные, а затем запросить содержание последнего файла снимка (.snap) мастера. Поскольку фактически в таком случае повторно проводится процесс настройки, это называется «повторная настройка». Тем не менее, есть отличие от обычной настройки – *идентификатор реплики* останется прежним. Если он изменится, то мастер посчитает, что в кластер добавляется новая реплика, и сохранит идентификатор экземпляра реплики, которой уже не существует. Полностью автоматизированный процесс повторной настройки появился в версии Tarantool 1.10.2.

## 9.9 Ограничения

**Количество частей в индексе**

Для TREE-индексов или HASH-индексов максимальное количество – 255 частей (`box.schema.INDEX_PART_MAX`). Для *RTREE-индексов* максимальное количество – 1, но это поля типа ARRAY (массив) с размерностью до 20. Для BITSET-индексов максимальное количество – 1.

**Количество индексов в спейсе**

128 (`box.schema.INDEX_MAX`).

**Количество полей в кортеже**

Теоретически максимальное количество составляет 2 147 483 647 полей (`box.schema.FIELD_MAX`). Практически максимальное количество указано в поле *field_count* спейса или соответствует максимальной длине кортежа.

**Количество байтов в кортеже**

Максимальное количество байтов в кортеже примерно равно *memtx_max_tuple_size* или *vinyl_max_tuple_size* (с ресурсами метаданных около 20 байтов на кортеж, которые добавляются к полезным байтам). Значение `memtx_max_tuple_size` или `vinyl_max_tuple_size` по умолчанию составляет 1 048 576. Чтобы его увеличить, укажите большее значение при запуске экземпляра Tarantool. Например, `box.cfg{memtx_max_tuple_size=2*1048576}`.

**Количество байтов в индекс-ключе**

Если поле в кортеже может содержать миллион байтов, то индекс-ключ может содержать миллион байтов, поэтому максимальное количество определяется такими факторами, как *количество байтов в кортеже*, а не параметрами индекса.

**Количество элементов в полях-массивах, входящих в индекс по массиву (multikey)**

In a Tarantool space that has *multikey indexes*, any tuple cannot contain more than ~8,000 elements in a field indexed with that multikey index. This is because every element has 4 bytes of metadata, and the tuple's metadata, which includes multikey metadata, cannot exceed 2^16 bytes.

**Количество спейсов**

Теоретически максимальное количество составляет 2 147 483 647 (`box.schema.SPACE_MAX`), но практически максимальное количество – около 65 000.

**Количество соединений**

Практически пределом является количество файловых дескрипторов, которые можно определить с операционной системой.

**Размер спейса**

The total maximum size for all spaces is in effect set by *memtx.memory*, which in turn is limited by the total available memory.

**Число операций обновления**

Максимальное количество операций, возможное в рамках одного обновления (для одного тапла), составляет 4000 (`BOX_UPDATE_OP_CNT_MAX`).

**Количество пользователей и ролей**

32 (`BOX_USER_MAX`).

**Длина имени индекса, имени спейса или имени пользователя**

65000 (`box.schema.NAME_MAX`).

**Количество реплик в наборе реплик**

32 (`vclock.VCLOCK_MAX`).

## 9.10 Рекомендации по Lua-синтаксису

В *функциях управления данными* Lua-синтаксис может различаться. Далее приводятся варианты таких различий на примере запросов `select()`. Аналогичные правила существуют и для остальных функций.

В каждом из приведенных примеров выполняются следующие действия: производится выборка по набору кортежей из спейса с именем „tester“, где значение поля, которое соответствует ключу в первичном индексе, равно 1. Также во всех примерах мы принимаем, что числовой идентификатор спейса „tester“ равен 512, но это верно только для нашей тестовой базы.

### 9.10.1 Способы ссылки на объект

Во-первых, есть три *способа ссылки на объект*:

```
-- #1 модуль.подмодуль.имя
tarantool> box.space.tester:select{1}
-- #2 заменить имя буквенной константой в квадратных скобках
tarantool> box.space['tester']:select{1}
-- #3 использовать переменную для всей ссылки на объект
tarantool> s = box.space.tester
tarantool> s:select{1}
```

Для примеров в документации, как правило, используется вариант синтаксиса №1, например «`box.space.`*`tester`*`:`». Но вы можете с тем же успехом пользоваться любым из трех описанных выше вариантов.

Также описания в руководстве используют синтаксис типа «`space_object:`» для ссылки на спейсы и «`index_object:`» для ссылки на индексы (например, `box.space.`*`tester`*`.index.`*`primary`*`:`).

### 9.10.2 Способы задания параметров

Затем есть семь *способов задания параметров*:

```
-- #1
tarantool> box.space.tester:select{1}
-- #2
tarantool> box.space.tester:select({1})
-- #3
tarantool> box.space.tester:select(1)
-- #4
tarantool> box.space.tester.select(box.space.tester,1)
-- #5
tarantool> box.space.tester:select({1},{iterator='EQ'})
-- #6
tarantool> variable = 1
tarantool> box.space.tester:select{variable}
-- #7
tarantool> variable = {1}
tarantool> box.space.tester:select(variable)
```

В Lua допускается пропуск круглых скобок () при вызове функции, если единственным аргументом является Lua-таблица, и иногда мы этим пользуемся в примерах. Вот почему `select{1}` аналогич-

но `select({1})`. Литеральные значения, такие как 1 (скалярное значение) или {1} (значение Lua-таблицы), можно заменить именами переменных, как в примерах 6 и 7.

Хотя есть особые случаи, когда фигурные скобки можно опустить, рекомендуется использовать их, потому что они означают Lua-таблицу. В примерах и описаниях данного руководства применяется форма {1}. Однако это тоже вопрос предпочтений пользователя, и на практике применимы все варианты.

### 9.10.3 Правила именования объектов

**Правила именования** объектов базы данных не слишком ограничены: максимальная длина составляет 65000 байтов (не символов), допускается практически любой символ Юникода, включая пробелы, идеограммы и знаки пунктуации.

В таких случаях во избежание путаницы с операторами и разделителями в Lua ссылки на объекты должны иметь форму типа литерал в квадратных скобках (2) или форму переменной (3). Например:

```
tarantool> box.space['1*A']:select{1}
tarantool> s = box.space['1*A !@$%^&*()_+12345678901234567890']
tarantool> s:select{1}
```

Не разрешаются:

- символы, которые представляют собой неназначенные кодовые точки,
- разделители строки и абзаца,
- управляющие символы,
- символ замены (U+FFFD).

Не рекомендуются: символы, которые не отображаются.

Имена зависимы от регистра, поэтому „A“ и „a“ – это не одно и то же.

ГЛАВА 10

# Contributing

## 10.1 How to get involved in Tarantool

### 10.1.1 What is Tarantool?

Tarantool is an open source database that can store everything in RAM. Use Tarantool as a cache with the ability to save data to disk. Tarantool serves up to a million requests per second, allows for secondary index searches, and has SQL support.

In Tarantool, you can execute code alongside data. This allows for faster operations. Implement any business logic in Lua. Get rid of stale entries, sync with other data sources, implement an HTTP service.

Go to *Getting Started* and try Tarantool.

### 10.1.2 How to get help?

We have a special Telegram chat for contributors. We speak Russian and English in the chat.

This is the easiest way to get your questions answered. Many people are afraid to ask questions because they believe they are «wasting the experts“ time,» but we don't really think so. Contributors are important to us.

We also have a Stack Overflow tag.

Join the chat and ask questions.

### 10.1.3 How to leave feedback, ideas, or suggestions?

You can leave your feedback or share ideas in different ways:

- **The simplest way** is to fill the feedback form. All you need to do is fill in one product comment field and click «Send.» You can optionally provide your email address. If you wish, we can involve you in the product development process.
- **A more technical way** is to create a ticket on GitHub. If you have a suggestion for a new feature or information about a bug, create a new GitHub issue. The link leads to the `tarantool/tarantool` repository. To leave feedback for our other projects on GitHub, select «Issues» > «New issue.»

See an example of a feature request.

To talk to our team about a product, go to one of our chats:

- Russian-speaking
- English-speaking

If Telegram is inconvenient for you or simply isn't working, you can leave your comment on tarantool.io. Fill out the form at the bottom of the site and leave your email. We read every request and respond to them usually within 2 days.

### 10.1.4 How to contribute

There are many ways to contribute to Tarantool:

- Code: Contribute to the code. We have components written in C, Lua, Python, Go, and other languages.
- Write: Improve documentation, write blog posts, create tutorials or solution pages.
- Q&A: Share your experience on Stack Overflow with the #tarantool tag.
- Spread the word: Share your accomplishments on social media using the `#tarantool` hashtag (or CC `@tarantooldb` on Twitter).

### 10.1.5 Tarantool ecosystem

Tarantool has a large ecosystem of tools. We divide the ecosystem into four large blocks:

- Tarantool itself.
- Modules for Tarantool. They can be written in C and Lua.
- Connectors for programming languages.
- Applied tools. See the curated Awesome Tarantool list, which also includes external tools.

To start contributing, check the «good first issue» tag in the issues section of any of our repositories. These are beginner to intermediate tasks that will help you get comfortable with the tool.

See the list of tasks for the `tarantool/tarantool` repository.

There is a review queue in each of our repositories, so your changes may not be reviewed immediately. We usually give the first answer within two days. Depending on the ticket and its complexity, the review time may take a week or more.

Please do not hesitate to tag the maintainer in your GitHub ticket.

Read on to learn about contributing to different ecosystem blocks.

### 10.1.6 Documentation: How to report and fix problems

There are several ways to improve the documentation:

- **The easiest one** is to leave your comment on the web documentation page. To use the built-in feedback form, select the text that you want to comment on, press `Ctrl+Enter`, type your comment in the pop-up window, and click *Submit*. On mobile screens, an *Error?* button appears at the bottom of the screen, which opens the same pop-up window. You can point out an error, provide feedback on the current article, or suggest changes. We review each comment and work with it.
- **Advanced**: All Tarantool documentation tasks can be found in the repository. Go to any task and suggest your changes. We write our documentation using reStructuredText markup, and we have a *writing style guide*. After you make the change, build the documentation locally and see how it works. This can be done automatically in Docker. To learn more, check the README of the tarantool/doc repository.

Some Tarantool projects have their documentation in the code repository. This is typical for modules, for example, metrics. This is done on purpose, so the developers themselves can update it faster. You can find instructions for building such documentation in the code repository.

If you find that the documentation provided in the README of a module or a connector is incomplete or wrong, the best way to influence this is to fix it yourself. Clone the repository, fix the bug, and suggest changes in a pull request. It will take you five minutes but it will help the whole community.

If you cannot fix it for any reason, create a ticket in the repository and report the error. It will be fixed promptly.

### 10.1.7 How to contribute to modules

Tarantool is a database with an embedded application server. This means you can write any code in C or Lua and pack it in distributable modules.

We have official and unofficial modules. Here are some of our official modules:

- HTTP server: HTTP server implementation with middleware support.
- queue: Tarantool implementation of the persistent message queue.
- metrics: Ready-to-use solution for collecting metrics.

Official modules are provided in our organization on GitHub.

All modules are distributed through our package manager, which is pre-installed with Tarantool. That also applies to unofficial modules, which means that other users can get your module easily.

If you want to add your module to our GitHub organization, send us a message on Telegram.

#### Contributing to an existing module

Tasks for contributors can be found in the issues section of any repository under the «good first issue» tag. These tasks are beginner or intermediate in terms of difficulty level, so you can comfortably get used to the module of your interest.

Check the currently open tasks for the HTTP Server module.

Please see our *Lua style guide*.

You can find the contact of the current maintainer in the MAINTAINERS file, located in the root of the repository. If there is no such file, please let us know. We will respond within two days.

If you see that the project does not have a maintainer or is inactive, you can become its maintainer yourself. See the *How to become a maintainer* section.

#### Creating a new module

You can also create custom modules and share them with the community. Look at the module template and write your own.

### 10.1.8 How to contribute to Tarantool Core

Tarantool is written mostly in C. Some parts are in C++ and Lua. Your contributions to Tarantool Core may take longer to review because we want the code to be reliable.

To start:

- *Learn how to build Tarantool*.
- Read about Tarantool architecture and main modules on the developer site and on GitHub.

In Tarantool development, we strive to follow the standards laid out in our *style and contribution guides*. These documents explain how to format your code and commits as well as how to write tests without breaking anything accidentally.

The guidelines also help you create patches that are easy to check, which allows quickly pushing changes to master.

Please read about our code review procedure before making your first commit.

You can suggest a patch using the fork and pull mechanism on GitHub: Make changes to your copy of the repository and submit it to us for review. Check the GitHub documentation to learn how to do it.

### 10.1.9 How to write tests

A database is a product that is expected to be as reliable as possible. We at Tarantool created `test-run`, a dedicated test framework for developing scripts that test Tarantool itself.

Writing your own test is not difficult. Check out the following examples:

- C unit test
- Lua unit test

We also have a CI workflow that automatically checks build and test coverage for new changes on all supported operating systems. The workflow is launched after every commit to the repository.

We have many tasks for QA specialists. Our QA team provides test coverage for our products, helps develop the test framework, and introduces and maintains new tools to test the stability of our releases.

For modules, we use luatest— our fork of a framework popular in the Lua community, enhanced and optimized for our tasks. See examples. of writing tests for a module.

### 10.1.10 How to contribute to language connectors

A connector is a library that provides an API to access Tarantool from a programming language. Tarantool uses its own binary protocol for access, and the connector's task is to transfer user requests to the database and application server in the required format.

Data access connectors have already been implemented for all major languages. If you want to write your own connector, you first need to familiarize yourself with the Tarantool binary protocol. Read *the protocol description* to learn more.

We consider the following connectors as references:

- `https://github.com/tarantool-php/client`
- net.box—Tarantool binary protocol client

You can look at them to understand how to do it right.

Some connectors in the Tarantool ecosystem are supported by the Tarantool team. Others are developed and supported exclusively by the community. All of them have their pros and cons. See the complete list of connectors and their recommended versions.

If you are using a community connector and want to implement new features for it or fix a bug, send your PRs via GitHub to the connector repository.

If you have questions for the author of the connector, check the MAINTAINERS file for the repository maintainer's contact. If there is no such file, send us a message on Telegram. We will help you figure it out. We usually answer within one day.

### 10.1.11 How to contribute to tools

The Tarantool ecosystem has tools that facilitate the workflow, help with application deployment, or allow working with Kubernetes.

Here are some of the tools created by the Tarantool team:

- tt: a CLI utility for creating and managing Tarantool applications.
- tarantool-operator: a Kubernetes operator for cluster orchestration.

These tools can be installed via standard package managers: `ansible galaxy`, `yum`, or `apt-get`.

If you have a tool that might go well in our curated Awesome Tarantool list, read the guide for contributors and submit a pull request.

### 10.1.12 How to become a maintainer

Maintainers are people who can merge PRs or commit to master. We expect maintainers to answer questions and tickets on time as well as do code reviews.

If you need to get a review but no one responds within a week, take a look at the Maintainers section of the repository's `README.md`. Write to the person listed there. If you have not received an answer within 3–4 days, you can escalate the question on Telegram.

A repository may have no maintainers (empty Maintainers list in `README.md`), or the existing maintainers may be inactive. In this case, you can become a maintainer yourself. We think it's better if the repository is maintained by a newbie than if the repository is dead. So don't be shy: we love maintainers and help them figure it all out.

All you need to do is fill out this form. Tell us what repository you want to access, the reason (inactivity, the maintainer is not responding), and how to contact you. We will consider your application in 1 day and either give you the rights or tell you what else needs to be done.

## 10.2 How to write release notes

Below are some best practices to make changelogs consistent, neat, and human-oriented.

### 10.2.1 Language

- Use the past tense to describe changed or fixed behavior.

  **Examples**

  Fixed false positive panic when yielding in debug hook (gh-5649).

  Fedora 32 is supported now. Added per-commit testing, updated YUM repositories (gh-4966).

- Use the present tense to describe new behavior.

  **Example**

  Data changes in read-only mode are now forbidden (gh-5231).

- Start with a capital letter, end with a period.

Note that these guidelines differ from the best practice for commit message titles that suggests using the imperative mood.

### 10.2.2 Formatting

In release notes, use the following Sphinx syntax when referring to a specific version of Tarantool:

```
Tarantool :tarantool-release:`2.10.0`.
This is a link to the release notes on GitHub.
```

The result looks like this:

> Tarantool v. 2.10.0. This is a link to the release notes on GitHub.

## 10.3 Building to contribute

To build Tarantool from source files, you need the following tools:

- Git
- GCC. Or Clang for Mac OS
- CMake 3.3 or later
- GNU Make

- Autoconf, any version
- Automake, any version
- Libtool, any version
- Readline, any version
- ncurses, any version
- OpenSSL, any version
- ICU, any version
- Zlib-devel, any version
- Python3 and modules:
  - pyyaml
  - gevent
  - six

### 10.3.1 Quick build

To install all required dependencies, build Tarantool and run tests, choose your OS and follow the instructions:

- *Ubuntu/Debian*
- *Fedora*
- *RHEL/CentOS 7*
- *CentOS 8*
- *Mac OS*
- *FreeBSD*

Some additional steps might be useful:

- *-DENABLE_DIST=ON for tarantoolctl installation*
- *Make RPM and Debian packages*
- *Verify your Tarantool installation*

#### Ubuntu/Debian

```
$ apt-get -y install git build-essential cmake autoconf automake libtool make \
  zlib1g-dev libreadline-dev libncurses5-dev libssl-dev libunwind-dev libicu-dev \
  python3 python3-yaml python3-six python3-gevent

$ git clone https://github.com/tarantool/tarantool.git --recursive

$ cd tarantool

$ git submodule update --init --recursive

$ make clean          # unnecessary, added for good luck
$ rm CMakeCache.txt   # unnecessary, added for good luck
```

(continues on next page)

(продолжение с предыдущей страницы)

```
$ mkdir build && cd build

$ # start initiating with build type=RelWithDebInfo
$ cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo

$ make

$ make test
```

### Fedora

```
$ dnf install -y git gcc gcc-c++ cmake  autoconf automake libtool make \
  readline-devel ncurses-devel openssl-devel zlib-devel libunwind-devel libicu-devel \
  python3-pyyaml python3-six python3-gevent

$ git clone https://github.com/tarantool/tarantool.git --recursive

$ cd tarantool

$ git submodule update --init --recursive

$ make clean         # unnecessary, added for good luck
$ rm CMakeCache.txt  # unnecessary, added for good luck

$ mkdir build && cd build

$ # start initiating with build type=RelWithDebInfo
$ cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo

$ make

$ make test
```

### RHEL/CentOS 7

```
$ yum install -y python-pip
$ yum install -y epel-release

$ curl -s https://packagecloud.io/install/repositories/packpack/backports/script.rpm.sh␣
↪| bash

$ yum install -y git gcc cmake3  autoconf automake libtool make gcc-c++ zlib-devel \
  readline-devel ncurses-devel openssl-devel libunwind-devel libicu-devel \
  python3-pyyaml python3-six python3-gevent

$ git clone https://github.com/tarantool/tarantool.git --recursive

$ cd tarantool
```

(continues on next page)

(продолжение с предыдущей страницы)

```
$ git submodule update --init --recursive

$ make clean         # unnecessary, added for good luck
$ rm CMakeCache.txt  # unnecessary, added for good luck

$ mkdir build && cd build

$ # start initiating with build type=RelWithDebInfo
$ cmake3 .. -DCMAKE_BUILD_TYPE=RelWithDebInfo

$ make

$ make test
```

### CentOS 8

```
$ dnf install -y epel-release

$ dnf install -y git gcc cmake3  autoconf automake libtool libarchive make gcc-c++ \
  zlib-devel readline-devel ncurses-devel openssl-devel libunwind-devel libicu-devel \
  python3-pyyaml python3-six python3-gevent

$ git clone https://github.com/tarantool/tarantool.git --recursive

$ cd tarantool

$ git submodule update --init --recursive

$ make clean         # unnecessary, added for good luck
$ rm CMakeCache.txt  # unnecessary, added for good luck

$ mkdir build && cd build

$ # start initiating with build type=RelWithDebInfo
$ cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo

$ make

$ make test
```

### Mac OS

This instruction is for those who use Homebrew. Refer to the full instruction for Mac OS if you use MacPorts.

```
$ xcode-select --install
$ xcode-select -switch /Applications/Xcode.app/Contents/Developer

$ git clone https://github.com/tarantool/tarantool.git --recursive

$ cd tarantool

$ git submodule update --init --recursive

$ brew install git openssl readline curl icu4c libiconv zlib cmake autoconf automake␣
↪libtool

$ pip install --user -r test-run/requirements.txt

$ make clean          # unnecessary, added for good luck
$ rm CMakeCache.txt   # unnecessary, added for good luck

$ mkdir build && cd build

$ # start initiating with build type=RelWithDebInfo
$ cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo

$ make

$ make test
```

### FreeBSD

```
$ git clone https://github.com/tarantool/tarantool.git --recursive

$ cd tarantool

$ git submodule update --init --recursive

$ pkg install -y git cmake autoconf automake libtool gmake readline icu

$ pip install --user -r test-run/requirements.txt

$ make clean          # unnecessary, added for good luck
$ rm CMakeCache.txt   # unnecessary, added for good luck

$ mkdir build && cd build

$ # start initiating with build type=RelWithDebInfo
$ cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo

$ gmake
```

(continues on next page)

(продолжение с предыдущей страницы)

```
$ gmake test
```

### 10.3.2 Additional steps

#### -DENABLE_DIST=ON for tarantoolctl installation

**Важно:** `tarantoolctl` is deprecated in favor of *tt CLI*. Find the instructions on switching from `tarantoolctl` to `tt` in *Migration from tarantoolctl to tt*.

The CMake option for hinting that the result will be distributed is `-DENABLE_DIST=ON`. With this option, `make install` installs `tarantoolctl` files in addition to `tarantool` files.

#### Make RPM and Debian packages

This step is optional. It's only for people who want to redistribute Tarantool. We highly recommend to use official packages from the tarantool.org web-site. However, you can build RPM and Debian packages using PackPack. Consult Build RPM or Deb package using packpack for details.

#### Verify your Tarantool installation

```
$ # если tarantool установлен локально после сборки
$ tarantool
$ # - ИЛИ -
$ # если tarantool не установлен локально после сборки
$ ./src/tarantool
```

Tarantool запустится в интерактивном режиме.

#### See also

- Tarantool README.md
- Building Tarantool on macOS
- Building Tarantool on FreeBSD
- Building Tarantool on OpenBSD
- Tarantool static build tooling

## 10.4 Добавление собственного модуля

На этой странице рассказано, как создать модуль для Tarantool, а затем разместить его на странице модулей Tarantool <http://tarantool.org/rocks.html>`_ и включить его в официальные образы Tarantool для Docker.

Чтобы узнать, как разработать несложный Lua-модуль для локального использования, прочитайте *практическое руководство*.

Чтобы помочь разработчикам, мы создали modulekit, набор шаблонов для создания Tarantool-модулей на Lua и C.

---

**Примечание:** Чтобы использовать `modulekit`, необходимо предварительно установить пакет `tarantool-dev`. Например, в Ubuntu выполните команду:

```
$ sudo apt-get install tarantool-dev
```

---

### 10.4.1 Добавление собственного модуля на Lua

Подробную информацию и примеры см. в README в ветке «luakit» репозитория tarantool/modulekit.

### 10.4.2 Добавление собственного модуля на C

В некоторых случаях может потребоваться создание Tarantool-модуля на C, а не на Lua, например, для работы со специальным оборудованием или низкоуровневыми системными интерфейсами.

Подробную информацию и примеры см. в README в ветке «ckit» репозитория tarantool/modulekit.

---

**Примечание:** Вы можете аналогичным образом создавать модули на C++ при условии, что в их коде не будут выбрасываться исключения.

---

## 10.5 Рекомендации

### 10.5.1 Рекомендации для разработчиков

#### Как работать над дефектами

На любой, даже незначительный дефект, меняющий видимое для пользователя поведение сервера, необходимо составить отчет об ошибке. Сообщите о дефекте на GitHub.

Когда вы сообщаете об ошибке, постарайтесь сразу же приступить к тестовому сценарию. Установите текущую контрольную точку для исправления ошибки и укажите серию. Назначьте задачу на себя. Укажите статус «In progress» (выполняется). Как только патч готов, укажите статус ошибки «In review» (на рассмотрении) и отправьте версию с исправленными ошибками на рассмотрение.

После успешного рассмотрения кода опубликуйте патч и укажите статус «Closed» (закрыт).

Патчи для исправления ошибок должны содержать ссылку на страницу нужной задачи на GitHub или на идентификатор задачи. Каждому патчу должен соответствовать отдельный тест. Если в текущем окружении создать подходящий тест затруднительно, следует предупредить тестировщиков.

Когда ваш патч доходит до главной ветки проекта, нужно сделать следующее:

- перевести статус ошибки в „fix committed“ (исправлено),
- удалить отдельную ветку.

### Как писать сообщение о коммите

Любой коммит следует описать в полезном сообщении. Следуйте нижеприведенным рекомендациям при коммитах в любой репозиторий Tarantool на GitHub.

1. Отделяйте тему от тела сообщения пустой строкой.
2. Постарайтесь ограничить тему сообщения примерно **50 символами**.
3. Начните тему сообщения с прописной буквы, если ей не предшествует префикс с именем подсистемы и точка с запятой:
   - memtx:
   - vinyl:
   - xlog:
   - replication:
   - recovery:
   - iproto:
   - net.box:
   - lua:
   - sql:
4. Не заканчивайте тему сообщения точкой.
5. Не пишите «gh-xx», «closes #xxx» в строке темы.
6. В теме сообщения используйте повелительное наклонение. Правильно оформленная тема Git-коммита должна корректно дополнять следующее предложение: «Если применить, коммит */здесь тема сообщения/*».
7. Уместите тело сообщения в примерно **72 символа**.
8. Используйте тело сообщения, чтобы объяснить, **что и почему**, а не как.
9. Привяжите задачи на GitHub в последних строках (см. как).
10. Используйте настоящие имя и адрес электронной почты. Членам проектной команды Tarantool рекомендуется указывать почту на **@tarantool.org**, но это необязательно.

Шаблон:

```
Кратко сформулируйте изменения в пределах 50 символов.

При необходимости, более подробные объяснения.
Уместите детали в примерно 72 символов.
Иногда первая строка считается темой
```

(continues on next page)

(продолжение с предыдущей страницы)

```
коммита, а остальной текст -- телом сообщения.
Критически важна пустая строка, которая отделяет тему от тела сообщения
(если только тело не отсутствует совсем); различные средства, такие как `log`,
`shortlog` и `rebase` могут их перепутать, если нет разделения.

Объясните проблему, которую решает данный коммит. Уделите внимание тому, почему
вы вносите эти изменения, а не как (это объясняется в коде).
Есть ли побочные эффекты или другие неочевидные последствия применения этих
изменений? Здесь можно объяснить их.

Следующие абзацы идут после пустых строк.

- Можно также использовать элементы в списке.

- Как правило, в качестве маркера применяется дефис или звездочка, которой предшествует
  пробел, а между строками вставляются пустые строки, но в данном случае
  условные обозначения могут разниться.

Fixes #123
Closes #456
Needed for #859
See also #343, #789
```

Некоторые реальные примеры:

- tarantool/tarantool@2993a75
- tarantool/tarantool@ccacba2
- tarantool/tarantool@386df3d
- tarantool/tarantool@076a842

Based on [1] and [2].

### 10.5.2 Documentation & Localization guidelines

These guidelines aim to help the team and external contributors write, translate, publish, and collaborate on the Tarantool documentation.

The guidelines are a work in progress, and we welcome all contributions.

#### Language and style

#### General style

#### Concise is good

People usually read technical documentation because they want something up and running quickly. Write simpler, more concise sentences.

Split the content into smaller paragraphs to improve readability. This will also eliminate the need for using `|br|` and help us translate content faster. Any paragraph over 6 sentences is large.

### Keep your audience in mind

Consider your audience's level. A getting started guide should be written in simpler terms than an advanced internals description.

If you choose to use metaphors to clarify a concept, make sure they are relatable for an international audience of IT professionals.

### Don't say «we»

Only use the pronoun «we» in entry-level texts like getting started guides. In other cases, avoid using «we», because it is unclear who that is exactly. Consider how Gentoo does it.

### Stick to the facts

Use measurable facts instead of personal judgments. Different users may have different ideas of what «often», «slow», or «small» means.

**Bad example:** This parameter is rarely updated.

**Good example:** This parameter is updated every two hours or more rarely.

### Refer to absolute time

Temporal adverbs like «today», «currently», «now», «in the future», etc. are relative – that is, they are based on the time the documentation is created.

Instead of these words, use absolute terms like version numbers or years. The meaning of those terms doesn't change over time.

If technical documentation is tied semantically to the time it was created, it increases the risk of the documentation becoming obsolete.

**Bad example:** Previously, the functionality worked differently. Currently, it supports SSL.

**Good example:** Before version x.y.z, the functionality worked differently. Since version x.y.z, it supports SSL.

### Express one idea in a sentence

Say exactly one thing in a sentence. If you want to define or clarify something, do it in a separate sentence. Simple sentences are easier to read, understand and translate.

| Don't | Do |
|---|---|
| Dogs (I have three of them) are my favorite animals. Their names are Ace, Bingo and Charm; Charm is the youngest one. | Dogs are my favorite animals. I have three of them. Their names are Ace, Bingo and Charm. Charm is the youngest one. |
| memtx (in-memory движок базы данных) используется по умолчанию, который был первым. | memtx is an in-memory storage engine. It is the default and was the first to arrive. |
| The replica set from where the bucket is being migrated is called the source; the target replica set where the bucket is being migrated to is called the destination. | The replica set from where the bucket is being migrated is called the source. The target replica set where the bucket is being migrated to is called the destination. |

### Put examples next to theory

It's best if examples immediately follow the concept they illustrate. The readers wouldn't want to look for the examples in a different part of the article.

### Specify link text

When you provide a *link*, clearly specify where it leads. In this way, you will not mislead the reader.

Bad example:

> For more details, click *here*.
>
> Use *this*.

Good example:

> For more details, refer to the documentation on *making links*.
>
> Use full *link names*.

## Formatting

### Use lists and tables

Lists and tables help split heavy content into manageable chunks.

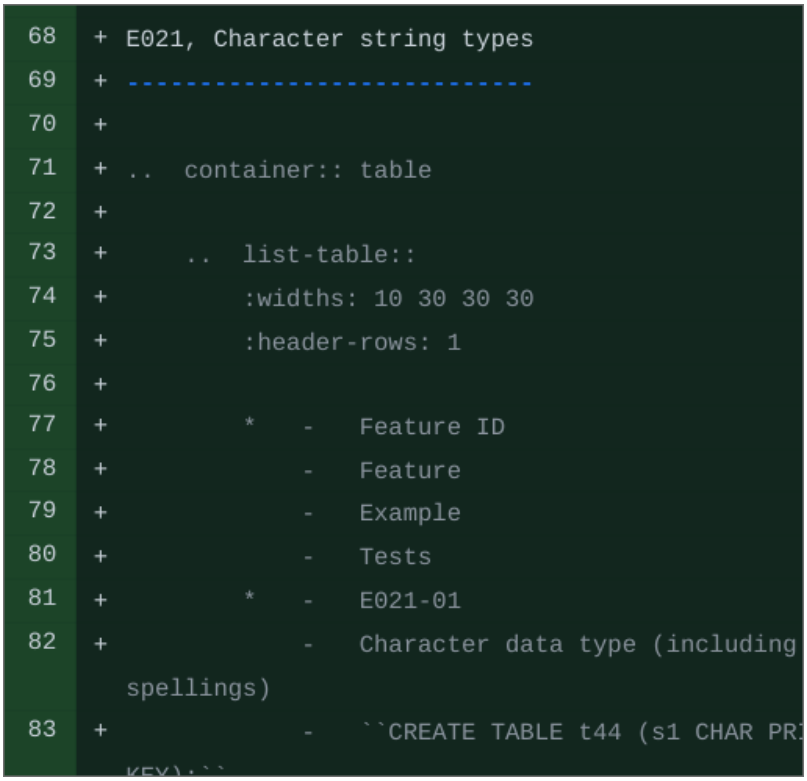
To make tables maintainable and easy to translate, use the `list-table` directive, as described in the Tarantool *table markup reference*.

Translators find it hard to work with content «drawn» with ASCII characters, because it requires adjusting the number of spaces and manually counting characters.

Bad example:

```
67 -     +------------+-----------------------------------
   ----------+-------------------------------------------
   --------------+---------------------------------------
   -----------------+
68 -     | E021-06    | SUBSTRING function.
   | ``SELECT substring(s1 FROM 1 FOR 1) FROM t;``
   | Fail. There is no such function. There is a function
   |
69 -     |            |
   |
   | :ref:`SUBSTR(x,n,n) <sql_function_substr>` which is
   |
70 -     |            |
   |
```

Good example:



```
+ E021, Character string types
+ ----------------------------
+
+ ..  container:: table
+
+     ..  list-table::
+         :widths: 10 30 30 30
+         :header-rows: 1
+
+         *   -   Feature ID
+             -   Feature
+             -   Example
+             -   Tests
+         *   -   E021-01
+             -   Character data type (including
  spellings)
+             -   ``CREATE TABLE t44 (s1 CHAR PR
  KEY);``
```

### Format code as code

Format large code fragments using the `code-block` directive, indicating the language. For `shorter code snippets`, make sure that only code goes in the backticks. Non-code shouldn't be formatted as code, because this confuses users (and translators, too). Check our guidelines on *writing about code*.

For more about formatting, check out the Tarantool *markup reference*.

### Word choice

#### Экземпляр или сервер

We say «instance» rather than «server» to refer to a Tarantool server instance. This keeps the manual terminology consistent with names like `/etc/tarantool/instances.enabled` in the Tarantool environment.

Wrong usage: «Replication allows multiple Tarantool *servers* to work with copies of the same database.»

Correct usage: «Replication allows multiple Tarantool *instances* to work with copies of the same database.»

#### Don't use i.e. and e.g.

Don't use the following contractions:

- «i.e.»—from the Latin «id est». Use «that is» or «which means» instead.
- «e.g.»—from the Latin «exempli gratia». Use «for example» or «such as» instead.

Many people, especially non-native English speakers, aren't familiar with the «i.e.» and «e.g.» contractions or don't know the difference between them. For this reason, it's best to avoid using them.

### Spelling and punctuation

#### Tarantool capitalization

The word «Tarantool» is capitalized because it's a product name. The only context where it can start with a lowercase «t» is code. Learn more about *code formatting in Tarantool documentation*.

#### Британский или американский вариант английского

Use the US English spelling.

#### Check your spelling and punctuation

Consider checking spelling, grammar, and punctuation with special tools like LanguageTool or Grammarly.

#### Dashes

Special symbols like dashes, quotation marks, and apostrophes look the same across all Tarantool documentation in a single language. This is because the documentation *builder* renders specific character sequences in the source into correct typographic characters.

Tarantool documentarians are recommended to use the **en dash** (–) only. Type two hyphens to insert it: `--`. Add spaces on both sides of the dash. Don't use a single hyphen as a dash.

Use the dash for the following purposes:

- To separate extra information.
- To mark a break in a sentence.
- To mark ranges like 4–16 GB (don't surround the dash with spaces in this case).

When indicating a range like `code element 1–code element 2`, escape the series of hyphens using character-level inline markup. Otherwise, the RST interpreter will perceive the dash as part of the RST syntax:

```
``box.begin()``\--``box.commit()``
```

### Ending punctuation in lists and tables

The following recommendations are *for the English language only*. You can find similar guidelines for the Russian language in the external reference for Russian proofreaders.

#### Lists

There are two kinds of lists:

- Where each item forms a complete sentence.
- Where each item is a phrase of three or less words or a term.

In the former case, start each item with a capital letter and end with a period. In the latter case, start it with a lowercase letter and add no ending punctuation (no period, no comma, no semicolon).

A list should be formatted uniformly: choose the first or second rule for all items in a list.

The above rules are adapted from the Microsoft style guide.

The sentence preceding a list can end either with a semicolon or a period.

Don't add redundant conjunctions like «and»/»or» before the last list item.

General English punctuation rules still apply for text in lists.

#### Tables

For the text in cells, use periods or other end punctuation only if the cells contain complete sentences or a mixture of fragments and sentences. (This is also a Microsoft guideline for the English language.)

Besides, make sure that your table punctuation is consistent – either all similar list/table items end with a period or they all don't. In the example below, *all* items in the second column don't have ending punctuation. Meanwhile, *all* items in the fourth column end with a period, because they are a mix of fragments and sentences:

| | | | |
|---|---|---|---|
| E011-04 | Arithmetic operators | `SELECT 10+1, 9-2, 8*3, 7/2 FROM t;` | Okay. |
| E011-05 | Numeric comparisons | `SELECT * FROM t WHERE 1 < 2;` | Okay. |
| E011-06 | Implicit casting among the numeric data types | `SELECT * FROM t WHERE s1 = 1.00;` | Okay, because Tarantool allows comparison of 1.00 with an INTEGER column. |

To learn more about table formatting, check the *table markup reference*.

### Localization

This section covers the specifics of localizing Tarantool into Russian. If you are translating Tarantool docs into Russian, be sure to check out our translation guidelines.

### State of localization

| Repository | State | Volume, words |
|---|---|---|
| Tarantool Community Edition | | 352 000 |
| Tarantool Ansible Role | | 11 000 |
| Tarantool Enterprise Edition | | 6 000 |
| Tarantool Data Grid | | 4 000 |
| Tarantool Metrics | | 4 000 |
| Tarantool C++ Driver | | 4 000 |
| Tarantool Kubernetes Operator | | 2 750 |
| Tarantool Luatest | | 750 |
| Tarantool Graphana Dashboard | | 500 |

### Glossaries

| Term [en] | Term [ru] | Description [en] |
|---|---|---|
| space | спейс | A space is a container for tuples. |
| tuple | кортеж | A tuple plays the same role as a “row” or a |
| Tarantool | Tarantool | НЕ ПЕРЕВОДИТЬ |
| primary index | первичный индекс | The first index defined on a space is called |
| fiber | файбер | A fiber is a set of instructions which are exe |
| Tarantool garbage collector | сборщик мусора в Tarantool | A garbage collector fiber runs in the backgr |
| Lua garbage collector | сборщик мусора на Lua | Lua manages memory automatically by run |
| storage engine | движок базы данных | A storage engine is a set of very-low-level ro |
| thread | поток | A thread of execution is the smallest sequer |
| Lua application | Lua-приложение, приложение на языке Lua | Tarantool’s native language for writing app |
| memtx | memtx | |
| instance | экземпляр | |
| implicit casting | неявное приведение типов | |
| database | база данных | |
| Release policy | Релизная политика | A set of rules for releasing and naming new |
| field | поле | Fields are distinct data values, contained in |
| leader election | выборы лидера | (in a replica set, by the Raft algorithm) |
| replica set | набор реплик | |
| heartbeat | контрольный сигнал | |
| functionality | функциональность | |
| log | журнал | |
| node | узел | |
| follower | реплика | |
| small allocator | аллокатор small | |
| patch | патч | |
| breaking change | критическое изменение | |
| parser | парсер | |

| Term [en] | Term [ru] | Description [en] |
|---|---|---|
| UUID | UUID | |
| data type | тип данных | |
| alias | алиас | |
| push | выполнить push | |
| MVCC | (механизм) MVCC | |
| dirty read | «грязное чтение» | |
| snapshot | снимок (данных) | |
| keywords | ключевые слова | |
| identifier | имя, идентификатор | |
| clause | предложение, блок | (SQL) A clause in SQL is a part of a query |
| expression | выражение | |
| predicate | предикат | (SQL) Predicates, which specify conditions |
| query | запрос | (SQL) Queries retrieve the data based on sp |
| result set | результат запроса | (SQL) An SQL result set is a set of rows fr |
| statement | инструкция | (SQL) A statement is any text that the dat |
| | | Tarantool: A statement consists of SQL-lan |
| batch | пакет (инструкций) | (SQL) A series of SQL statements sent to t |
| production configuration | конфигурация производственной среды | |
| deployment | развертывание | Transforming a mechanical, electrical, or co |
| roll back | отменить | |
| deploy to production | | IT infrastructure deployment typically invo |
| operations | эксплуатация | (DevOps) Information technology operation |
| to deploy | | Transforming a mechanical, electrical, or co |
| deployment plan | | A sequence of operations or steps that must |
| production environment | производственная среда | Production environment is a term used mos |
| failover | восстановление после сбоев | In computing and related technologies such |
| directory | директория | |
| bucket | сегмент | |
| select | выберите, выбрать | To select a checkbox |

### Localization guidelines

Use this guide when localizing Tarantool into Russian.

### Tone of voice

### General voice

We address IT specialists fairly knowledgeable in their respective fields. The goal of our translations is to help these people understand how to use Tarantool. Think of us as their colleagues and address them as such. Be professional but friendly. Don't command or patronize. Use colloquial speech but avoid being too familiar. It's all about the golden mean.

### Modal verbs

Think twice when translating modal verbs. Avoid using expressions like «вы должны», because they sound like a demand in Russian, and «вам придётся», because it implies that our readers will face a lot of trouble. Be careful with «нельзя». Make it easy for the user to read the documentation.

### Gender neutrality

Use gender-neutral expressions like «сделать самостоятельно» instead of «сделать самому», etc.

### Term choice

Though not all of our readers may be fluent in English, they write in English-based programming languages and are used to seeing error messages in English. Therefore, if they see an unfamiliar and/or more archaic Russian term for a familiar concept, they might have trouble correlating them.

We don't want our audience to feel confused, so we prefer newer terms. We also provide the English equivalent for a term if it is used in the article for the first time.

If you feel like an older Russian term may sound more familiar for a part of the audience (for example, those with a math background), consider adding it in parentheses along with the English equivalent. Don't repeat the parentheses throughout the text. A similar rule applies to *introducing terms in Tarantool documentation*.

### Term choice examples

| | First time | All following times |
|---|---|---|
| state machine | машина состояний (конечный автомат, state machine) | машина состояний |
| write-ahead log; WAL | журнал упреждающей записи (write-ahead log, WAL) | журнал упреждающей записи; WAL; журнал WAL *(using a descriptor)* |

### Best practices

### Be creative

Please avoid word-for-word translations. Let the resulting text sound as though it was originally written in Russian.

### Less is more

Be concise and don't repeat yourself. Fewer words are the best option most of the time.

| Don't | Do |
|---|---|
| Профиль доступа можно назначить для любой роли пользователя, созданной администратором. А к ролям по умолчанию привязать профили доступа не получится, поскольку такие роли редактировать нельзя. | Профиль доступа можно назначить для любой роли пользователя, созданной администратором. Исключение составляют роли по умолчанию, поскольку их нельзя редактировать. |

### Topic and focus

Avoid English word order.

The Russian speech is structured with topic and focus (тема и рема). The topic is the given in the sentence, something we already know. The focus is whatever new/important information is provided in the sentence about the topic. In written Russian, the focus most often stands at the end of the sentence, while in English, sentences may start with it.

| | |
|---|---|
| It is recommended to use `systemd` for managing the application instances and accessing log entries. | Для управления экземплярами приложения и доступа к записям журнала рекомендуется использовать `systemd`. |
| Do not specify working directories of the instances in this configuration. | Не указывайте в этой конфигурации рабочие директории экземпляров. |

### No bureaucratese

Avoid overly formal, bureaucratic language whenever possible. Prefer verbs over verbal nouns, and don't use «являться» and «осуществляться» unless it's absolutely necessary.

To learn how to clear your Russian texts of bureaucratese, check Timur Anikin's training «The Writing Dead».

| Don't | Do |
|---|---|
| Сообщение исчезнет, как только вы покинете данную страницу. | Сообщение исчезнет, как только вы покинете страницу. |
| Проверка истечения срока действия паролей производится раз в 30 минут. | Раз в 30 минут система проверяет, не истек ли срок действия паролей. |

### Consistency

Use one term for one concept throughout the article. For example, only translate production as «производственная среда» and not as «эксплуатационная среда» throughout your article. It's not about synonyms, but about terms: we don't want people to get confused.

#### Avoid elliptical sentences

| | Don't | Do |
|---|---|---|
| Defaults to `root`. | По умолчанию — `root`. | Значение по умолчанию — `root`. |

#### Pronoun collocations

Do all the pronouns point to the exact nouns you want them to?

**Example (how not to):** Прежде чем добавить запись в конфигурацию, укажите к ней путь.

In the example, it is not quite clear what «к ней» means – to the record or to the configuration. For more on this issue, check out the writers" reference at «Ошибкариум».

#### Be critical towards your text

Don't forget to proofread your translation. Check your text at least twice.

#### Be nice to your peers

If you review others' translations, be gentle and kind. Everyone makes mistakes, and nobody likes to be punished for them. You can use phrasings like «I suggest» or «it's a good idea to... .»

### Defining and using terms

#### What are concepts and terms

To write well about a certain subject matter, one needs to know its details and use the right, carefully selected words for them. These details are called concepts, and the words for them are called terms.

**concept** A concept is the idea of an object, attribute, or action. It is independent of languages, audience, and products. It just exists.

For example, a large database can be partitioned into smaller instances. Those instances are easier to operate, and their throughput often exceeds the throughput of a single large database instance. The instances can exchange data to keep it consistent between them.

**term** A term is a word explicitly selected by the authors of a particular text to denote a *concept* in a particular language for a particular audience.

For example, in Tarantool, we use the term «[database] sharding» to denote the concept described in the previous example.

### Use preferred terms

The purpose of using terms is writing concisely and unambiguously, which is good for the readers. But selecting terms is hard. Often, the community favors two or more terms for one concept, so there's no obvious choice. Selecting and consistently using any of them is much better than not making a choice and using a random term every time. This is why it's also helpful to restrict the usage of some terms explicitly.

**restricted term** A restricted term is a word that the authors explicitly prohibited to use for denoting a *concept*. Such a word is sometimes used as a *term* for the same *concept* elsewhere – in the community, in books, or in other product documentation. Sometimes this word is used to denote a similar but different concept. In this case, the right choice of terms helps us differentiate between concepts.

For example, in Tarantool, we don't use the term «[database] segmentation» to denote what we call «database sharding.» Nevertheless, other authors might do so. We also use the term «[database] partitioning» to denote a wider concept, which includes sharding among other things.

### Define terms by explaining concepts

We always want to document definitions for the most important concepts, as well as for concepts unique to Tarantool.

Define every term in the document that you find most appropriate for it. You don't have to create a dedicated glossary page containing all the definitions.

To define a term, use the `glossary` directive in the following way:

```
..  glossary::

    term
        definition text

    term2
        definition text
```

There can be several `glossary` directives in a Sphinx documentation project and even in a single document. This page has two of them, for example.

The Sphinx documentation has an extensive glossary that can be used as a reference.

### Introduce terms on first entry

When you use a term in a document for the first time, define it and provide synonyms, a translation, examples, and/or links. It will help readers learn the term and understand the concept behind it.

1. Define the term or give a link to the definition.

   Database sharding is a type of horizontal partitioning.

   To give a link to the definition, use the `term` role:

   ```
   For example, this is a link to the definition of :term:`concept`.
   Like any rST role, it can have :term:`custom text <concept>`.
   ```

   The resulting output will look like this:

   For example, this is a link to the definition of *concept*. Like any rST role, it can have *custom text*.

With acronyms, you can also use the `abbr` role:

```
Delete the corresponding :abbr:`PVC (persistent volume claim)`...
```

It produces a tooltip link: PVC (persistent volume claim).

2. Provide synonyms, including the *restricted terms*. Only do it on the first entry of a term.

   Database sharding (also known as . . . ) is a type of. . .

3. When writing in Russian, it's good to add the corresponding English term. Readers may be more familiar with it or can search it online.

   Шардирование (сегментирование, sharding) — это. . .

4. Give examples or links to extra reading where you can.

### Markup reference

Tarantool documentation is built via the Sphinx engine and is written in reStructuredText. This section will guide you through our typical documentation formatting cases.

### General syntax guidelines

#### Basic syntax

Paragraphs contain text and may contain inline markup: *emphasis*, **strong emphasis**, `interpreted text`, `inline literals`.

Text can be organized in bullet-lists:

```
*   This is a bullet list.

*   Bullets can be "*", "+", or "-".

    -   Lists can be nested. And it is good to indent them with 4 spaces.
```

or in enumerated lists:

```
1.  This is an enumerated list.

2.  Tarantool build uses only arabic numbers as enumerators.

#.  You can put #. instead of point numbers and Sphinx will
    recognize it as an enumerated list.
```

### Перенос текста

It's good practice to wrap lines in documentation source text. It makes source better readable and results in lesser `git diff`'s. The recommended limit is 80 characters per line for plain text.

In new documents, try to wrap lines by sentences, or by parts of a complex sentence. Don't wrap formatted text if it affects rST readability and/or HTML output. However, wrapping with proper indentation shouldn't break things.

### Отступы

In rST, indents play exactly the same role as in Python: they denote object boundaries and nesting.

For example, a list starts with a marker, then come some spaces and text. From there, all lines relating to that list item must be at the same indentation level. We can continue the list item by creating a second paragraph in it. To do that we have to leave it at the same level.

We can put a new object inside: another list, or a block of code. Then we have to indent 4 more spaces.

It's best if all indents are multiples of 4 spaces, even in lists. Otherwise the document is not consistent. Also, it is much easier to put indents with tabs than manually.

Note that you have to use two or three spaces instead of one. It is allowed in rST markup:

```
|...|...|...|...
*   unordered list
#.  ordered list
..  directive::
|...|...|...|...
```

Пример:

```
|...|...|...|...
#.  List item 1.
    Paragraph continues.

    Second paragraph.

#.  List item 2.

    *   Nested list item.

        ..  code-block:: bash

            # this code block is in a nested list item

    *   Another nested list item.
|...|...|...|...
```

Resulting output:

1. List item 1. Paragraph continues.

   Second paragraph.

2. List item 2.

   - Nested list item.

```
# this code block is in a nested list item
```

- Another nested list item.

### Добавление комментариев

Sometimes we may need to leave comments in an rST file. To make Sphinx ignore some text during processing, use the following per-line notation with `.. //` as the comment marker:

```
.. // здесь комментарий
```

The starting characters `.. //` do not interfere with the other rST markup, and they are easy to find both visually and using `grep`. To find comments in source files, go ahead with something like this:

```
$ grep -n "\.\. //" doc/reference/**/*.rst
doc/reference/reference_lua/box.rst:47:.. // moved to "User Guide > 5. Server␣
↪administration":
doc/reference/reference_lua/box.rst:48:.. // /book/box/triggers
...
```

If you're working with PyCharm or other similar IDE, links in the console will be clickable and will lead right to the source file and string. Check it out!

```
doc/reference/reference_lua/box.rst:47:.. // moved to "User Guide > 5. Server administration":
doc/reference/reference_lua/box.rst:48:.. // /book/box/triggers
```

These comments don't work properly in nested documentation, though. For example, if you leave a comment in module -> object -> method, Sphinx ignores the comment and all nested content that follows in the method description.

### Headings

### Heading markup

We use the following markup for headings:

```
Level 1 heading
===============

Level 2 heading
---------------

Level 3 heading
~~~~~~~~~~~~~~~

Level 4 heading
^^^^^^^^^^^^^^^
```

The underlining should be exactly the same length as the heading text above it. Mismatching length will result in a build warning.

Sphinx allows using other characters and styles to format headings. Indeed, using this markup consistently helps us better reuse and move content. It also helps us recognize the heading level immediately without reading the whole document and calculating levels.

If you're going to make a 4th or 5th level heading, you probably need to split the document instead.

#### Title headings

The top-level heading of each document plays the important role of a document title. Title's text is used in several places:

- Literally as a `<h1>` tag in HTML or top-level heading in other formats.
- Text in the breadcrumbs — the path to the document shown above the text.
- The `:doc:` link's *default text*.
- Part of the page's `title` tag contents, used as the browser tab name.

```
<title>
    Documentation guidelines | Tarantool
</title>
```

- Potentially, the page's OpenGraph metadata which is used for building page preview cards on social networks and messengers.

```
<meta property="og:title" content="Documentation guidelines">
```

ard to navigate in a hierarchy of more than three heading levels.

### Links and references

#### Linking to other documentation pages

To create a link to another document in our documentation, we use the `:doc:` role. For example, this link points to the document `/reference/reference_lua/box_error.rst`:

```
:doc:`box.error reference </reference/reference_lua/box_error>`
```

Our convention is to put the full path to the referred document so that we can easily replace the path if it changes. Note that we can omit the `.rst` part of the filename.

You can use the target document's title as the link text. To do so, omit the text in the link definition:

```
:doc:`/reference/reference_lua/box_index`
```

And you will get this:

*Вложенный модуль box.index*

### Linking to labels (anchors)

To generate a link to the certain place in the page, we use the `:ref:` role. For this purpose, we add our own labels for linking to any place in this documentation.

Соглашение об именовании заключается в следующем:

- Character set: a through z, 0 through 9, hyphen, underscore.
- Format: `path hyphen filename hyphen tag`

**Пример:**

```
..  _c_api-box_index-iterator_type:
```

где:

- `c_api` is the directory name,
- `box_index` is the file name (without «.rst»), and
- `iterator_type` is the tag.

Use a hyphen «-» to delimit the path and the file name. In the documentation source, we use only underscores «_» in paths and file names, reserving the hyphen «-» as the delimiter for local links.

The tag can be anything meaningful. The only guideline is for Tarantool syntax items (such as members), where the preferred tag syntax is `module_or_object_name hyphen member_name`. For example, `box_space-drop`.

To add a link to an anchor, use the following syntax:

```
Check out the :ref:`Quick start guide <vshard-quick-start>`.
```

The result will be like this:

> Check out the *Quick start guide*.

### Ссылки на внешние ресурсы

To make an external link, use the following syntax:

```
Feel free to report an issue at `Tarantool GitHub <https://github.com/
↪tarantool/tarantool/issues>`_.
```

Избегайте разделения ссылки и определения цели, например:

```
Feel free to report an issue at `Tarantool GitHub`_.

..  _Tarantool GitHub: https://github.com/tarantool/tarantool/issues
```

because every separated link tends to cause troubles when this documentation is translated to other languages.

### Tables

Tables are very useful and rST markup offers different ways to create them.

We prefer list-tables because they allow you to put as much content as you need without painting ASCII-style borders:

```
..  container:: table

    ..  list-table::
        :widths: 25 75
        :header-rows: 1

        *   -   Name
            -   Use

        *   -   :doc:`/reference/reference_lua/box_ctl/wait_ro`
            -   Wait until ``box.info.ro`` is true
```

This is how the table will look like:

| Имя | Назначение |
|---|---|
| *box.ctl.wait_ro()* | Дождаться, пока не будет выполнено `box.info.ro` |

Notice that we use `*` and then `-` in tables because it is more readable when rows and columns marked differently.

### Writing about code

When writing articles, you need to format code specially, separating it from other text. This document will guide you through typical cases when it is recommended to use code highlighting.

### Defining what code is

In general, code is any text, processed by a machine. It is also probably code if the expression contains characters that ordinary words do not have, such as `_`, `{}`, `[ ]`, `..`. Also, you should format the expression as code if it fits at least one of the items in the list below:

- parts of a programming language: names of classes, variables, and functions, short expressions, data types and so on,
- multiline fragments of application logs,
- example link which the reader will not open: `example.com`, `https://example.com:80`,
- parts of URL, like port number,
- package names,
- CLI app names.

Items we don't format as code:

- names of products, organizations and services, for example, Tarantool, memtx, vinyl
- well-established terms such as stdin and stdout

Keep in mind that grammar doesn't apply to code, even inline.

- Correct: «use `shellcheck` to analyze your Bash code».
- Incorrect: «`shellcheck` your Bash code». Please do not use code as a verb.
- Even worse: «shellcheck your Bash code». There's no such word in English and we don't explain what to use.
- Cursed: «try `shellchecking` your Bash code». There's no such word and no such application.

### Code blocks and inline code

If you have to choose between inline code and code block highlighting, pay attention to the following guidelines:

### Code snippets

Use code blocks when you have to highlight multiple lines of code. Also, use it if your code snippet contains a standalone element that is not a part of the article's text.

For code snippets, we use the `code-block:: language` directive. You can enable syntax highlighting if you specify the language for the snippet. The most commonly used highlighting languages are:

- `tarantoolsession` – interactive Tarantool session, where command lines start with `tarantool>` prompt.
- `console` – interactive console session, where command lines start with `$` or `#`.
- `lua`, `bash` or `c` for programming languages.
- `text` for cases when we want the code block to have no highlighting.

Sphinx uses the Pygments library for highlighting source code. For a complete list of possible languages, see the list of Pygments lexers.

For example, a code snippet in Lua:

```
..  code-block:: lua

    for page in paged_iter("X", 10) do
      print("New Page. Number Of Tuples = " .. #page)
      for i=1,#page,1 do print(page[i]) end
    end
```

Lua syntax is highlighted in the output:

```
for page in paged_iter("X", 10) do
  print("New Page. Number Of Tuples = " .. #page)
  for i=1,#page,1 do print(page[i]) end
end
```

Note that in code blocks you can write comments and translate them:

```
..  //Here is the first comment.
..  //Here is the second comment.
```

### Inline code

Use inline code when you need to wrap a short snippet of code in text, such as variable name or function definition. Keep in mind that inline code doesn't have syntax highlighting.

To format some inline text as `code`, enclose it with double ` characters or use the `:code:` role:

```
*   Formatting code with backticks: ``echo "Hello world!"``.

*   Formatting code with a role: :code:`echo "Hello world!"`.
```

Both options produce the same output:

- Formatting code with backticks: `echo "Hello world!"`.
- Formatting code with a role: `echo "Hello world!"`.

### Notes on using inline-code

- If you have expressions such as `id==4`, you should format the whole expression as code inline. Also, you can use the words «equals», «doesn't equal» or other similar words without formatting expression as code. Both variants are correct.
- Inline code can be used to highlight expressions that are hard to read, for example, words containing `il`, `Il` or `OO`.

### Highlighting variables in code

If you need to mark up a placeholder inside code inline, use the `:samp:` or our custom `:extsamp:` role, like this:

```
:samp:`{space_object}:insert(\\{ffi.cast('double', {value})\\})`

:extsamp:`{*{space_object}*}:insert({ffi.cast('double', {**{value}**})})`
```

And you will get this:

*space_object*:insert({ffi.cast('double', *value*)})

*space_object*:insert({ffi.cast('double', value)})

Notice two backslashes before the curly brackets in the first line. They are needed to escape curly brackets from Lua syntax.

As you can see, `:extsamp:` extends the abilities of `:samp:`. It allows you to highlight placeholders in both italics and bold and avoid escaping curly brackets. `:extsamp:` has the following syntax:

- `{*{element}*}` for *italic*
- `{**{element}**}` for **bold**

If you need to mark up a placeholder in code block, use the following syntax:

```
..  cssclass:: highlight
..  parsed-literal::

    :samp:`box.space.{space-name}:create_index('{index-name}')`
```

The output will look like this:

box.space.*space-name*:create_index('*index-name*')

### Formatting file and directory names

If you need to highlight some file standalone name or path to file in text, use the `:file:` role. You can use curly braces inside this role to mark up a replaceable part:

```
:file:`/usr/bin/example.py`

:file:`/usr/{dirname}/example.py`

:file:`/usr/{dirname}/{filename.ext}`
```

And you will get this:

/usr/bin/example.py

/usr/*dirname*/example.py

/usr/*dirname*/*filename.ext*

### Referring to GUI elements

To mention a GUI element, use the `:guilabel:` directive:

```
Click the :guilabel:`OK` button.
```

### Admonitions

Sometimes you need to highlight a piece of information. For this purpose we use admonitions.

In Tarantool we have 3 variants of css-style for admonitions:

- Note:

```
..  note::
```

**Примечание:** This is a note. We use it to highlight extra information that might be helpful for users.

For example, *here* we provide a user with extra information about using `net_box.new()` function.

- Warning:

```
..  warning::
```

**Предупреждение:** This is a warning. As you might guess, we use it to warn users about something.

For example, in the description of *box.session.on_connect()* trigger we warn a user about some consequences of his actions.

- Important:

```
..  important::
```

---

**Важно:** This block contains essential information that the user should know while doing something.

For example, the block *in Getting Started with Docker* is used to warn the user against closing the terminal window.

---

- Custom admonition:

```
..  admonition:: Your title
    :class: fact
```

---

**Your title**

This is a fact. `fact` is our custom CSS class. Use it when neither note nor warning doesn't fit.

Note that this type requires a title.

For example, *here* we highlight the rules that are necessary to read, and that's why we use `fact`.

---

The docutils documentation offers many more variants for admonitions, but for now these three are enough for us. If you think that it is time to create the new style for some of these types, feel free to contribute or contact us to create a task.

### Documenting the API

This document contains general guidelines for describing the Tarantool API, as well as examples and templates.

### Style

Please write as simply as possible. Describe functionality using short sentences in the present simple tense. A short sentence consists of no more than two clauses. Consider using LanguageTool or Grammarly to check your English. For more style-related specifics, consult the *Language and style* section.

### Indicating the version

For every new module, function, or method, specify the version it first appears in.

For a new parameter, specify the version it first appears in if this parameter is a «feature» and the version it's been introduced in differs from the version introducing the function/method and all other parameters.

To specify the version, use the following Sphinx directive:

```
Since :doc:`2.10.0 </release/2.10.0>`.
This is a link to the release notes on the Tarantool documentation website.
```

The result looks like this:

> Since Tarantool *2.10.0*. This is a link to the release notes on the Tarantool documentation website.

### Language of the general description

Use one of the two options:

- Start with a verb in the imperative mood. Example: *Create a fiber*.
- Start with a noun. Example: *The directory where memtx stores snapshot files*.

### Checklist

Each list item is a characteristic to be described. Some items can be optional.

### Function or method

- *Since which Tarantool version*
- *General description*
- *Parameters*
- What this function returns (if nothing, write „none“)
- Return type (if exists)
- *Possible errors* (if exist)
- *Complexity factors* (for *CRUD operations* and *index access functions*)
- Usage with memtx and vinyl (if differs)
- Example(s)
- Extra information (if needed)

See *module function example*, *class method example*.

### Data

- *Since which Tarantool version*
- *General description*
- Return type
- Example

See *class data example*.

### Function and method parameters

- *Since which Tarantool version* (if added later)
- *General description*
- Type
- Default value (if optional), possible values

If the parameter is optional, make sure it is enclosed in square brackets in the function declaration (in the «heading»). Do not mark parameters additionaly as «optional» or «required»:

```
..  function:: format(URI-components-table[, include-password])

    Construct a URI from components.

    :param URI-components-table: a series of ``name:value`` pairs, one for each component
    :param include-password: boolean. If this is supplied and is ``true``, then
                             the password component is rendered in clear text,
                             otherwise it is omitted.
```

### Configuration parameters

Configuration parameters are not to be confused with class and method parameters. Configuration parameters are passed to Tarantool via the command line or in an initialization file. You can find a list of Tarantool configuration parameters in the *configuration reference*.

- *Since which Tarantool version*
- *General description*
- Type
- Corresponding environment variable (if applicable)
- Default value
- Possible values (can be included in the general description, for example, as a list)
- Dynamic (yes or no)

See *configuration parameter example*.

### Documenting possible errors

In the «Possible errors» section of a function or class method, consider explaining what happens if any parameter hasn't been defined or has the wrong value.

### Examples and templates

### Module functions

We use the Sphinx directives `.. module::` and `.. function::` to describe functions of Tarantool modules:

```
..  module:: fiber

..  function:: create(function [, function-arguments])

    Create and start a fiber. The fiber is created and begins to run immediately.

    :param function: the function to be associated with the fiber
    :param function-arguments: what will be passed to function.

    :return: created fiber object
    :rtype: userdata
```

(continues on next page)

(продолжение с предыдущей страницы)

```
**Example:**

..  code-block:: tarantoolsession

    tarantool> fiber = require('fiber')
    ---
    ...
    tarantool> function function_name()
             >   print("I'm a fiber")
             > end
    ---
    ...
    tarantool> fiber_object = fiber.create(function_name); print("Fiber started")
    I'm a fiber
    Fiber started
    ---
    ...
```

The resulting output looks like this:

`fiber.create`(*function*[, *function-arguments*])

Create and start a fiber. The fiber is created and begins to run immediately.

**Параметры**

- `function` – the function to be associated with the fiber
- `function-arguments` – what will be passed to function.

**Return** created fiber object

**Rtype** userdata

**Example:**

```
tarantool> fiber = require('fiber')
---
...
tarantool> function function_name()
         >   print("I'm a fiber")
         > end
---
...
tarantool> fiber_object = fiber.create(function_name); print("Fiber started")
I'm a fiber
Fiber started
---
...
```

### Class methods and data

Methods are described similarly to functions, but the `.. class::` directive, unlike `.. module::`, requires nesting.

As for data, it's enough to write the description, the return type, and an example.

Here is the example documentation describing the method and data of the `index_object` class:

```
..  class:: index_object

    ..  method:: get(key)

        Search for a tuple :ref:`via the given index <box_index-note>`.

        :param index_object index_object: :ref:`object reference
                                           <app_server-object_reference>`
        :param scalar/table      key: values to be matched against the index key

        :return: the tuple whose index-key fields are equal to the passed key values
        :rtype:  tuple

        **Possible errors:**

        * No such index
        * Wrong type
        * More than one tuple matches

        **Complexity factors:** index size, index type.
        See also :ref:`space_object:get() <box_space-get>`.

        **Example:**

        ..  code-block:: tarantoolsession

            tarantool> box.space.tester.index.primary:get(2)
            ---
            - [2, 'Music']
            ...

    ..  data:: unique

        True if the index is unique, false if the index is not unique.

        :rtype: boolean

        ..  code-block:: tarantoolsession

            tarantool> box.space.tester.index.primary.unique
            ---
            - true
            ...
```

And the resulting output looks like this:

object index_object

index_object:get(*key*)
Search for a tuple *via the given index*.

**Параметры**

- index_object (*index_object*) – *object reference*
- key (scalar/table) – values to be matched against the index key

**Return** the tuple whose index-key fields are equal to the passed key values

**Rtype** tuple

**Possible errors:**

- No such index
- Wrong type
- More than one tuple matches

**Complexity factors:** index size, index type. See also *space_object:get()*.

**Example:**

```
tarantool> box.space.tester.index.primary:get(2)
---
- [2, 'Music']
...
```

index_object.unique
True if the index is unique, false if the index is not unique.

**Rtype** boolean

```
tarantool> box.space.tester.index.primary.unique
---
- true
...
```

### Configuration parameters

Example:

```
    In that situation, queries may time out after ``vinyl_timeout`` seconds.
    This should rarely occur, since normally vinyl
    would throttle inserts when it is running low on compaction bandwidth.
    Compaction can also be ordered manually with
    :doc:`/reference/reference_lua/box_index/compact`.

    |
    | Type: float
    | Default: 60
    | Environment variable: TT_VINYL_TIMEOUT
    | Dynamic: yes
```

(continues on next page)

(продолжение с предыдущей страницы)

```
.. _cfg_basic-username:
```

Result:

This should rarely occur, since normally vinyl would throttle inserts when it is running low on compaction bandwidth. Compaction can also be ordered manually with *index_ object:compact()*.

Type: float
Default: 60
Environment variable: TT_VINYL_TIMEOUT
Dynamic: yes

### Images

Images are useful in explanations of concepts and structures. When you introduce a term or describe a structure of multiple interconnected parts (such as a cluster), consider illustrating it with a diagram. If you are explaining how to use a GUI, check if a screenshot can make the doc clearer.

Note that illustrations should complement the text, not replace it. Even with an image, the text should be enough for readers to understand the topic.

Don't overuse images: they are harder to support than text. Use them only if they bring an obvious benefit.

### Diagrams

There is a basic set of diagram elements – blocks, arrows, and other – to use in Tarantool docs. It is stored in this Miro board. It also provides basic rules for creating diagrams.

### Size

There are two sizes of diagram elements:

- **M** – bigger elements to use in diagrams with a small number of elements.
- **S** – smaller elements to use in diagrams with a big number of elements.

Avoid changing the size of diagram elements unless it's absolutely necessary.

The diagrams should have the same width. This guarantees that their elements have the same size on pages. The examples in the Miro board have frames of the right width. Copy the frame and and place your diagram in it without changing the frame width.

### Exporting

To save the diagram to a file:

1. Make the frame transparent so that it isn't shown in the resulting image (set its color to «no color»).
2. Select all elements together with the frame and click **Copy as image** in the context menu (under the three dots). The image will be copied to the clipboard.
3. Paste the image from the clipboard to any graphic editor, for example, GIMP.
4. Remove the Miro logo in the bottom right corner.
5. Export/save the image to PNG.

### Screenshots

Take screenshots with any tool you like.

Ensure screenshot consistency on the page:

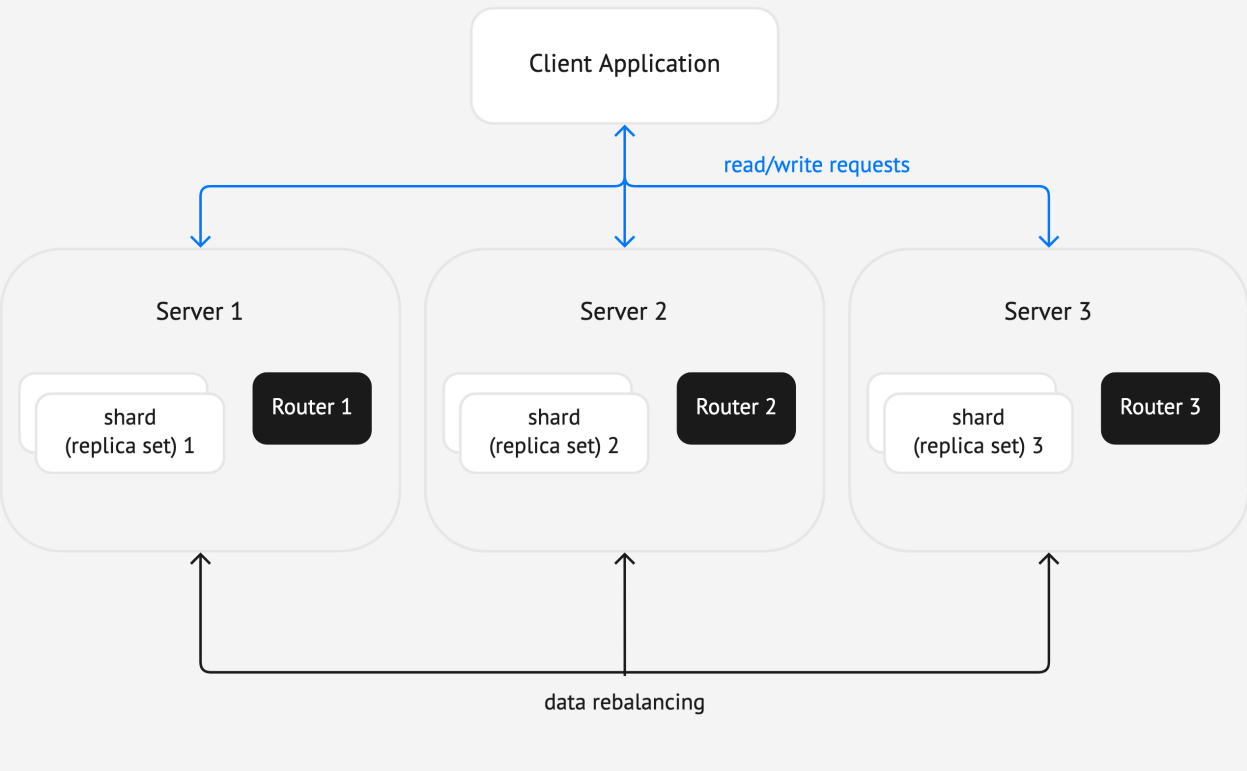
- Screenshots must show the same environment: operating system, product version, visual theme, and so on.
- The configuration and data must be consistent. For example, if you've shown spaces with data on a screenshot, subsequent screenshots must have the same data, too.
- Size and resolution must be the same across the page unless you want to zoom in to a specific part of the screen.

### Markup

Insert the images using the `image` directive:

```
..  image:: images/example_diagram.png
    :alt: Example diagram alt text
```

Result:




### Building Tarantool Docs

#### How to build Tarantool documentation using Docker

See Docker

#### Prepare for work

First of all, pull the image for building the docs.

```
docker pull tarantool/doc-builder:fat-4.3
```

Next, initialize a Makefile for your OS:

```
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "cmake ."
```

#### Update submodules and generate documentation sources from code

A big part of documentation sources comes from several other projects, connected as Git submodules. To include their latest contents in the docs, run these two steps.

1. Update the submodules:

```
git submodule update --init
git fetch --recurse-submodules
git submodule update --remote --checkout
```

This will initialize Git submodules and update them to the top of the stable branch in each repository.

`git submodule update` can sometimes fail, for example, when you have changes in submodules" files. You can reinitialize submodules to fix the problem.

**Caution:** all untracked changes in submodules will be lost!

```
git submodule deinit -f .
git submodule update --init
```

Note that there's an option to update submodule repositories with a `make` command. However, it's intended for use in a CI environment and not on a local machine.

```
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make pull-
↪modules"
```

2. Build the submodules content:

```
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make build-
↪modules"
```

This command will do two things:

1. Generate documentation source files from the source code
2. Copy these files to the right places under the `./doc/` directory.

If you're editing submodules locally, repeat this step to view the updated results.

Now you're ready to build and preview the documentation locally.

### Build and run the documentation on your machine

When editing the documentation, you can set up a live-reload server. It will build your documentation and serve it on 127.0.0.1:8000. Every time you make changes in the source files, it will rebuild the docs and refresh the browser page.

```
docker run --rm -it -p 8000:8000 -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c
↪"make autobuild"
```

First build will take some time. When it's done, open 127.0.0.1:8000 in the browser. Now when you make changes, they will be rebuilt in a few seconds, and the browser tab with preview will reload automatically.

You can also build the docs manually with `make html`, and then serve them using python3 built-in server:

```
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make html"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make html-ru"
python3 -m http.server --directory output/html
```

or python2 built-in server:

```
cd output/html
python -m SimpleHTTPServer
```

then go to localhost:8000 in your browser.

There are other commands which can run in the `tarantool/doc-builder` container:

```
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make html"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make html-ru"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make singlehtml"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make singlehtml-
↪ru"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make pdf"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make pdf-ru"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make json"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make json-ru"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make epub"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make epub-ru"
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make update-po"
```

#### Linkcheck

There's a specific build mode which checks internal and external links instead of producing a document.

```
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make linkcheck"
```

If you need to save the linkcheck's report in a file, you can use the following trick:

```
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make linkcheck"␣
↪2>&1 | tee linkcheck.log
```

Here `2>&1` redirects the `stderr` output to `stdout`, and then `tee` both shows in on screen and writes to a file.

#### Vale

Tarantool documentation uses the Vale linter for checking grammar, style, and word usage. Its configuration is placed in the `vale.ini` file located in the root project directory.

To enable RST support in Vale, you need to install Sphinx. Then, you can enable Vale integration in your IDE, for example:

- VS Code
- IntelliJ-based IDEs

#### Localization

Terms:

- **translation unit** (TU) is an atomic piece of text which can be translated. A paragraph, a list item, a heading, image's alt-text and so on.
- **translation source files** are the files with translation units in English only. They're located in `locale/en`.
- **translation files** are the files which match original text to translated text. They're located in `locale/ru`.

To update the translation files, run the `make update-po` task:

```
docker run --rm -it -v $(pwd):/doc tarantool/doc-builder:fat-4.3 sh -c "make update-po"
```

Translate the strings in the updated files and then commit the changes.

### How to contribute

To contribute to documentation, use the REST format for drafting and submit your updates as a pull request via GitHub.

To comply with the writing and formatting style, use the guidelines provided in the documentation, common sense and existing documents.

Notes:

- If you suggest creating a new documentation section (a whole new page), it has to be saved to the relevant section at GitHub.
- If you want to contribute to localizing this documentation (for example, into Russian), add your translation strings to `.po` files stored in the corresponding locale directory (for example, `/locale/ru/LC_MESSAGES/` for Russian). See more about localizing with Sphinx at `http://www.sphinx-doc.org/en/stable/intl.html`.

### Sphinx-build warnings reference

This document will guide you through the warnings that can be raised by Sphinx while building the docs.

Below are the most frequent warnings and the ways to solve them.

#### Bullet list ends without a blank line; unexpected unindent

**Similar warning:** Block quote ends without a blank line; unexpected unindent

**Пример:**

```
*   The last point of bullet list
This text should start after a blank line
```

**Решение:**

```
*   The last point of bullet list

This text should start after a blank line
```

#### Could not lex literal_block as «...». Highlighting skipped

This warning means that there's a `code-block` with an unknown lexer. Most probably, it's a typo. Check out the full list of Pygments lexers for the right spelling.

**Пример:**

```
..  code-block:: cxx

    // some code here
```

**Решение:**

```
..  code-block:: cpp

    // some code here
```

However, sometimes there's no appropriate lexer or the code snippet can't be lexed properly. In that case, use `code-block:: text`.

#### Duplicate explicit target name: «...»

**Пример:**

```
*   `Install <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>`_
    ``git``, the version control system.

*   `Install <https://linuxize.com/post/how-to-unzip-files-in-linux/>`_
    the ``unzip`` utility.
```

**Решение:**

Sphinx-builder raises warnings when we call different targets the same name. Sphinx developers recommend using double underlines `__` in such cases to avoid this.

```
*   `Install <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>`__
    ``git``, the version control system.

*   `Install <https://linuxize.com/post/how-to-unzip-files-in-linux/>`__
    the ``unzip`` utility.
```

#### Document isn't included in any toctree

This warning means that you forgot to put the document name in the toctree.

**Решение:**

If you don't want to include the document in a toctree, place the `:orphan:` directive at the top of the file. If this file is already included somewhere or reused, add it to the _includes directory. Sphinx ignores everything in this directory because we list it among `exclude_patterns` in `conf.py`.

#### Duplicate label «...», other instance in «.../.../...»

This happens if you include the contents of a file into another file, when the included file has tags in it. In this, Sphinx thinks the tags are repeated.

**Решение:**

As in the previous case, add the file to _includes or avoid using tags in it.

### Malformed hyperlink target

**Similar warning:** Unknown target name: «...»

Check the target spelling and the tag syntax.

**Пример:**

```
..  _box_space-index_func
```

```
See the :ref:`Creating a functional index <box_space-index_func>` section.
```

**Решение:**

A semicolon is missing in the tag definition:

```
..  _box_space-index_func:
```

### Anonymous hyperlink mismatch

**Warning example:** Anonymous hyperlink mismatch: 1 references but 0 targets.

Check the hyperlink formatting.

**Пример:**

```
Read more in `Lua Manual <https://www.lua.org/manual/5.3`__.
```

**Решение:**

A closing greater-than sign is missing in the tag definition:

```
Read more in `Lua Manual <https://www.lua.org/manual/5.3>`__.
```

### Toctree contains reference to nonexisting document „...“

**Пример:**

This may happen when you refer to a wrong path to a document.

**Решение:**

Check the path.

If the path points to a submodule, check that you've *built the submodules content* before building docs.

#### Undefined label: ... (if the link has no caption the label must precede a section header)

**Пример:**

```
Read more in :ref:`<sql_data_type_conversion>`.
```

**Решение:**

We recommend using custom captions with `:ref:`:

```
Read more in :ref:`Data Type Conversion <sql_data_type_conversion>`.
```

**См. также:**

- *Links and references*

#### Unexpected indentation

The reStructuredText syntax is based on indentation, much like in Python. All lines in a block of content must be equally indented. An increase or decrease in indentation denotes the end of the current block and the beginning of a new one.

**Пример:**

Note: In the following examples, dots stand for indentation spaces. For example, `|..|` denotes a two-space indentation.

```
|..|* (Engines) Improve dump start/stop logging. When initiating memory dump, print
how much memory is going to be dumped, the expected dump rate, ETA, and the recent
write rate.
```

**Решение:**

```
*|...|(Engines) Improve dump start/stop logging. When initiating memory dump, print
|....|how much memory is going to be dumped, the expected dump rate, ETA, and the recent
|....|write rate.
```

**См. также:**

- *General syntax guidelines*

#### Unknown document

**Пример:**

```
:doc:`reference/reference_lua/box_space/update`
```

**Решение:**

Sphinx did not recognize the file path correctly due to a missing slash at the beginning, so let's just put it there:

```
:doc:`/reference/reference_lua/box_space/update`
```

### Documentation infrastructure

This section of the *documentation guidelines* discusses some of the support activities that ensure the correct building of documentation.

### Adding submodules

The documentation source files are mainly stored in the documentation repository. However, in some cases, they are stored in the repositories of other Tarantool-related products or modules, such as Monitoring.

If you are working with source files from a product or module repository, add that repository as a submodule to the documentation repository and configure other necessary settings. This will ensure that the entire body of Tarantool documentation, presented on the official website, is built properly.

Here is how to do that:


- *1. Add a submodule*
- *2. Update build_submodules.sh*
- *3. Update .gitignore*


#### 1. Add a submodule

First, we need to add the repository with content source files as a submodule.

1. Make sure you are in the root directory of the documentation repository.
2. In the `./modules` directory, add the new submodule:

```
cd modules
git submodule add https://<path_to_submodule_repository>
cd ..
```

3. Check that the new submodule is in the `.gitmodules` file, for example:

```
[submodule "modules/metrics"]
   path = modules/metrics
   url = https://github.com/tarantool/metrics.git
```

#### 2. Update build_submodules.sh

Now define what directories and files are to be copied from the submodule repository to the documentation repository before building documentation. These settings are defined in the `build_submodules.sh` file in the root directory of the documentation repository.

Here are some real submodule examples that show the logic of the settings.

#### metrics

The content source files for the `metrics` submodule are in the `./doc/monitoring` directory of the submodule repository. In the final documentation view, the content should appear in the Monitoring chapter (`https://www.tarantool.io/en/doc/latest/book/monitoring/`).

To make this work:

- Create a directory at `./doc/book/monitoring/`.
- Copy the entire content of the `./modules/metrics/doc/monitoring/` directory to `./doc/book/monitoring/`.

Here are the corresponding lines in `build_submodules.sh`:

```
monitoring_root="${project_root}/modules/metrics/doc/monitoring" #
monitoring_dest="${project_root}/doc/book"

mkdir -p "${monitoring_dest}"
yes | cp -rf "${monitoring_root}" "${monitoring_dest}/"
```

The `${project_root}` variable is defined earlier in the file as `project_root=$(pwd)`. This is because the documentation build has to start from the documentation repository root directory.

### 3. Update .gitignore

Finally, add paths to the copied directories and files to `.gitignore`.

## Git workflow

### Branching

Use one branch for a single task, unless you're fixing typos or markup on several pages. Long commit histories are hard to manage and sometimes end up stale.

Start a new branch from the last commit on `latest`. Make sure to update your local version of `latest` with `git pull`. Otherwise, you may have to rebase later.

Name your branch so it's clear what you're doing. Examples:

- `short-issue-description`
- `gh-1234-short-issue-description`
- `your-github-handle/short-issue-description`

---

**Важно:** It is not recommended to submit PRs to the documentation repository from forks. Because of a GitHub failsafe mechanism, it is impossible to view changes from a fork on the development website.

Creating branches directly in the repository results in a more convenient workflow.

---

### Linking issues and PRs

When a PR is linked to an issue:

- You can go from the issue straight to the PR by clicking the link in the right column.
- The issue will be automatically closed when you close the PR.

Specify the issue(s) you want to close in the description of your PR. GitHub will connect them if you use specific keywords. Here are some of them:

- Closes #1234
- Resolves #1234
- Fixes #1234

If your PR closes more than one issue, mention each of them:

```
Resolves #1300, resolves #1234, resolves tarantool/doc#100
```

### Commit messages

- Most of the time, one-line commit messages are sufficient for documentation changes.
  - When you squash commits at merge, the resulting commit message is a sum of all commit messages in the PR. It is advised to include the «resolves» string in the first commit. Otherwise, there's a risk that this line won't be included in the merge commit.
- Convey the nature of the change and possibly the reason why it was made.
  - Don't specify the files you've changed or the issue you're working on. The file names can be looked up in the «Files» section of the PR, and the PR description has the issue number(s).
- Try keeping the commit title 50 characters or shorter.
- Use the imperative mood.
- Start with a capital letter, don't add ending punctuation.
- (Optional) Use the telegraphic style, or «headlinese», dropping the articles.

### Good examples

- `git commit -m "Expand section on msgpack"`
- `git commit -m "Add details on IPROTO_BALLOT"`
- `git commit -m "Create new structure"`
- `git commit -m "Improve grammar"`

#### Bad examples

- `git commit -m "Fix gh-2007, second commit"`
- `git commit -m “Changed the file box_protocol.rst”`
- `git commit -m "added more list items"`

#### Selecting a reviewer

Ideally, a PR should have two reviewers: a subject matter expert (SME) and a documentarian. The SME checks the facts, and the documentarian checks the language and style.

If you’re not sure who the SME for an issue is, try the following:

- Check the issue description. The SME is often mentioned there explicitly.
- Note who created the issue and who was involved in the discussion.

#### Merging

Merge when your document is ready and good enough. For external contributors, merging is blocked until a reviewer’s approval.

- Always squash commits.
- Make sure the commit message mentions all relevant issues with «resolves» or «fixes».
- Make sure you’ve attributed all participants with `Co-authored-by`.

### 10.5.3 Руководство по написанию кода на C

Для управления версиями мы используем Git. Разработки ведутся в ветке, используемой по умолчанию (сейчас `master`). Наш Git-репозиторий находится на GitHub, его можно выгрузить с помощью `git clone git://github.com/tarantool/tarantool.git` (анонимный пользователь получит доступ только для чтения).

Если у вас есть вопросы о внутреннем устройстве Tarantool, задайте их на StackOverflow или напрямую разработчикам Tarantool в Telegram.

**Общие рекомендации**

Стиль разработки проекта основан на стиле программирования ядра Linux.

Кроме того, мы даем дополнительные рекомендации, которые либо специфичны для Tarantool, либо отличаются от рекомендаций по программированию ядра Linux. Ниже приведен стиль программирования ядра Linux, переработанный с учетом особенностей разработки Tarantool.

### Стиль программирования Tarantool

В этом документе описан приоритетный стиль программирования Tarantool для разработчиков и участников сообщества. Мы **настаиваем** на соблюдении этих правил, чтобы наш код был последовательным и понятным любому разработчику.

### Глава 1: Отступы

Табуляция составляет 8 символов (8 символов табуляции, а не 8 пробелов), то есть отступы будут также составлять 8 символов. Появляются отступники, которые призывают делать отступы в 4 (или даже 2!) символа, а это сродни попытке округлить число Пи до 3.

Обоснование: Основная идея отступов состоит в том, чтобы показать, где начинается и заканчивается логический блок кода. Особенно если вы смотрите на один и тот же код в течение 20 часов, трудно не заметить пользу больших отступов.

Некоторые могут возразить, что отступ в 8 символов делает код слишком широким, особенно на 80-знаковой строке терминала. Ответ: Если вам понадобилось более трех уровней отступа, вы что-то делаете неправильно, и вам следует переписать этот участок.

Отступы в 8 символов облегчают чтение кода. Кроме того, по ним можно понять, когда у вас становится слишком много вложенных функций.

Лучше всего упростить несколько уровней отступов в операторе switch, выравнивая `switch` и его вспомогательные метки `case` в одном столбце вместо того, чтобы использовать двойные отступы для меток `case`, например:

```
switch (suffix) {
case 'G':
case 'g':
  mem <<= 30;
  break;
case 'M':
case 'm':
  mem <<= 20;
  break;
case 'K':
case 'k':
  mem <<= 10;
  /* fall through */
default:
  break;
}
```

Не размещайте несколько операторов на одной строке, если вам нечего скрывать:

```
if (condition) do_this;
  do_something_everytime;
```

И не размещайте несколько операторов присваивания на одной строке. Избегайте сложных выражений.

Пробелы используются только в комментариях и документации, но никогда — для отступов, и приведенный выше пример сломан намеренно.

Найдите достойный редактор и не оставляйте пробелы в конце строки.

### Глава 2: Разрыв длинных строк

Смысл стиля программирования заключается в читаемости и удобстве сопровождения с использованием общедоступных средств.

Длина строк ограничена 80 символами, и этому следует уделить особое внимание. Для комментариев установлен тот же лимит в 80 символов.

Операторы длиной более 80 символов будут разбиты на логические части. Можно сделать исключение, если это значительно повысит читаемость и не скроет информацию. Последующие части значительно короче основной и сильно смещены вправо. То же относится к заголовкам функций с длинным списком аргументов.

### Глава 3: Фигурные скобки и пробелы

Другая проблемой, которая всегда возникает в программировании на C, — размещение фигурных скобок. В отличие от отступов, есть несколько технических обоснований, чтобы выбрать один способ, а не другой, но всё же предпочтительно, как нам показали великие Керниган и Ричи, поместить открывающую скобку в конце строки, а закрывающую в начале новой строки:

```
if (x is true) {
  we do y
}
```

Это применимо ко всем блокам операторов без функций (if, switch, for, while, do), например:

```
switch (action) {
case KOBJ_ADD:
  return "add";
case KOBJ_REMOVE:
  return "remove";
case KOBJ_CHANGE:
  return "change";
default:
  return NULL;
}
```

И только в особенных случаях, а именно для функций, открывающая скобка размещается в начале следующей строки:

```
int
function(int x)
{
  body of function
}
```

Отступники по всему миру утверждали, что такая несогласованность . . . ну . . . несогласованна, но все здравомыслящие люди знают: (а) K&R **правы**, (б) K&R правы. Кроме того, функции в любом случае будут особенными (в C их нельзя вложить).

Обратите внимание, что за закрывающей скобкой на отдельной строке ничего нет; **кроме** тех случаев, когда за ней следует продолжение того же оператора, то есть `while` в do-операторе или `else` в if-операторе, например:

```
do {
  body of do-loop
} while (condition);
```

и

```
if (x == y) {
  ..
} else if (x > y) {
  ...
} else {
  ....
}
```

Обоснование: K&R.

Кроме того, обратите внимание, что такое расположение скобок также сводит к минимуму количество пустых (или почти пустых) строк без потери читаемости. Таким образом, поскольку новые строки на экране — это не возобновляемый ресурс (вспомним о 25-строчных экранах терминала), у вас будет больше пустых строк для комментариев.

Не используйте лишние фигурные скобки, если нужен всего один оператор.

```
if (condition)
  action();
```

и

```
if (condition)
  do_this();
else
  do_that();
```

Это не применимо, если только одна ветка условного оператора — это отдельный оператор. В последнем случае используйте фигурные скобки в обеих ветках:

```
if (condition) {
  do_this();
  do_that();
} else {
  otherwise();
}
```

### Глава 3.1: Пробелы

В том, что касается пробелов, стиль программирования Tarantool зависит (в основном) от использования функции или ключевого слова. Используйте пробел после (большинства) ключевых слов. Значимые исключения: `sizeof`, `typeof`, `alignof` и `__attribute__`, — которые похожи на функции (и обычно используются с круглыми скобками, хотя они и не обязательны, как в объявлении `sizeof info` после `struct fileinfo info;`).

Итак, вставляйте пробелы после этих ключевых слов:

```
if, switch, case, for, do, while
```

но не после `sizeof`, `typeof`, `alignof` или `__attribute__`. Например:

```
s = sizeof(struct file);
```

Не добавляйте пробелы вокруг (внутри) выражений в круглых скобках. Этот пример **неправильный**:

```
s = sizeof( struct file );
```

Объявляя данных типа указателя или функцию, которая возвращает тип указателя, лучше использовать `*` рядом с именем данных или именем функции, а не рядом с именем типа. Примеры:

```
char *linux_banner;
unsigned long long memparse(char *ptr, char **retptr);
char *match_strdup(substring_t *s);
```

Добавляйте по одному пробелу вокруг (с каждой стороны) большинства знаков двухместных и трехместных операций, например, любое из следующих:

```
=  +  -  <  >  *  /  %  |  &  ^  <=  >=  ==  !=  ?  :
```

но не добавляйте пробелы после знаков одноместных операций:

```
&  *  +  -  ~  !  sizeof  typeof  alignof  __attribute__  defined
```

не нужны пробелы перед знаками одноместных операций увеличения или уменьшения постфикса:

```
++  --
```

не нужны пробелы после знаков одноместных операций увеличения или уменьшения префикса:

```
++  --
```

и не нужны пробелы вокруг знаков элементов структуры `.` и `->`.

Не отделяйте оператор приведения от аргумента пробелом, например `(ssize_t)inj->iparam`.

Не оставляйте пробелы на концах строк. Некоторые редакторы с `smart` отступом вставляют пробелы в начале новых строк, поэтому вы можете сразу ввести следующую строку кода. Однако некоторые такие редакторы не удаляют пробелы, если вы не пишете там код, например, если вы оставите пустую строку. В результате имеем строки с пробелами в конце.

Git предупредит, если патчи содержат пробелы в конце строк, и может по желанию удалить пробелы за вас; однако, в серии патчей, это может привести к тому, что последующие патчи в серии не применятся, поскольку изменены контекстные строки.

### Глава 4: Именование

C — это спартанский язык, и именование должно быть спартанским. В отличие от разработчиков на Modula-2 и Pascal, разработчики на языке C не используют забавные имена, такие как ThisVariableIsATemporaryCounter. Разработчик на языке C назвал бы такую переменную `tmp`, что намного легче написать и не сложнее понять.

ОДНАКО, хотя на имена со смешанным регистром смотрят неодобрительно, обязательным требованием будут описательные имена глобальных переменных. Назвать глобальную функцию `foo` — это оскорбление.

У ГЛОБАЛЬНЫХ переменных (которые надо использовать, только если без них **нельзя** обойтись) должны быть описательные имена, равно как и у глобальных функций. Если у вас есть функция, которая подсчитывает количество активных пользователей, нужно назвать ее `count_active_users()` или как-то похоже, **не** стоит называть ее `cntusr()`.

Кодирование типа функции в названии (так называемая венгерская нотация) — это признак плохого тона, поскольку компилятор в любом случае знает типы и может их проверять, и это только путает программиста. Неудивительно, что MicroSoft делает глючные программы.

Имена ЛОКАЛЬНЫХ переменных должны быть короткими и точными. Если у вас есть счетчик случайных целых чисел, его следует называть `i`. Назвать его `loop_counter` непродуктивно, если нет никаких шансов, что его перепутают. Аналогично `tmp` может быть практически любой переменной, которая используется для хранения временного значения.

Если вы боитесь перепутать имена своих локальных переменных, у вас другая проблема, которая называется синдромом дисбаланса гормона роста функций. См. Главу 6 (Функции).

Для именования функций у нас есть такое правило:

- `new`/`delete` для функций, которые выделяют + инициализируют и удаляют + освобождают объект,
- `create`/`destroy` для функций, которые инициализируют/удаляют объект, но не занимаются управлением памятью,
- `init`/`free` для функций, которые инициализируют/удаляют библиотеки и подсистемы.

### Глава 5: Директива Typedef

Не используйте что-то вроде `vps_t`. Будет **ошибкой** использовать typedef для определения структур и указателей. Если вы видите в исходном коде

```
vps_t a;
```

что это означает? И наоборот, если говорится

```
struct virtual_container *a;
```

можно действительно понять, что такое `a`.

Многие думают, что typedef `способствует читаемости`. Это не так. Эту директиву нужно использовать для:

1. Непрозрачных объектов (где typedef активно используется для **сокрытия** объекта).

   Пример: `pte_t` и другие непрозрачные объекты, доступ к которым можно получить с помощью соответствующих функций доступа.

---

**Примечание:** Непрозрачность и `функции доступа` сами по себе не слишком хороши. Мы используем их для pte_t и т. п., потому что на самом деле там **нет** никакой информации для скачивания.

---

2. Явные целочисленные типы, где абстракция **помогает** не перепутать, `int` это или `long`.

   u8/u16/u32 — вполне нормальные typedef, хотя они больше подходят для пункта 4.

   ---

   **Примечание:** Опять же — для этого должна быть **причина**. Если есть «unsigned long», нет причины вводить typedef unsigned long myflags_t;

   ---

   но если есть четкая причина, почему при определенных обстоятельствах может быть `unsigned int`, а в других случаях может быть `unsigned long`, то на здоровье — используйте typedef.

3. Когда вы используете разрыв, чтобы буквально создать **новый** тип для проверки типов.

4. Новые типы, идентичные стандартным типам C99, в определенных исключительных обстоятельствах.

   Хотя глазам и мозгу нужно лишь короткое время, чтобы привыкнуть к стандартным типам, например `uint32_t`, некоторые в любом случае возражают против их использования.

   При редактировании существующего кода, в котором уже используется один или другой набор типов, следует придерживаться выбранного типа.

Возможно, есть и другие случаи, но основное правило состоит в следующем: НИКОГДА НЕ используйте typedef, если вы не соблюдаете одно из этих правил.

В общем, указатель или структура, содержащие элементы, к которым можно получить прямой доступ, **никогда** не должны быть typedef.

### Глава 6: Функции

Функции должны быть короткими и приятными, и выполнять только одно действие. Они должны помещаться на одном или двух экранах текста (размер экрана ISO/ANSI 80x24, как мы все знаем) и выполнять одно действие, но делать это хорошо.

Максимальная длина функции обратно пропорциональна сложности функции и уровню отступов. Итак, если у вас есть концептуально простая функция, которая представляет собой лишь один длинный (но простой) оператор вариант case, где вам нужно делать много мелочей для множества разных случаев, длинная функция — это нормально.

Однако, если у вас есть сложная функция, и вы подозреваете, что не слишком одаренный старшеклассник может даже не понять, о чем эта функция, следует придерживаться ограничений. Используйте вспомогательные функции с описательными именами (можно попросить компилятор встроить их, если считаете, что это критически важно для производительности, и он, вероятно, справится лучше).

Другим критерием функции является количество локальных переменных. Их не должно быть больше 5-10, или вы делаете что-то неправильно. Продумайте функцию заново и разбейте ее на более мелкие части. Человеческий мозг обычно легко отслеживает около 7 разных вещей, а больше — и он уже запутается. Вы знаете, что сейчас вы гений, но, возможно, через пару недель вам захочется понять, что именно вы делали.

В прототипах функций включайте имена параметров с типами данных. Хотя для языка C это и не требуется, но рекомендуется для Tarantool, потому что это простой способ добавить ценную информацию для читателя.

Обратите внимание, что тип возвращаемого значения функции располагается перед именем и сигнатурой функции.

### Глава 7: Централизованный выход из функции

Хотя некоторые объявили аналог оператора goto устаревшим, его часто используют компиляторы в виде инструкции безусловной передачи управления.

Оператор goto пригодится, когда функция производит выход из нескольких мест и необходимо выполнить какие-то общие действия, такие как очистка. Если очистка не нужна, пусть функция возвращается напрямую.

Выбирайте имена меток, которые объясняют, что делает goto или почему. Пример хорошего имени: `out_free_buffer:`, если goto освобождает `буфер`. Избегайте таких имен из GW-BASIC, как `err1:` и `err2:`, поскольку вам придется перенумеровать их, если вы будете добавлять или удалять пути выхода, и в любом случае они затрудняют проверку.

Обоснование использования goto:

- безусловные операторы легче понять и выполнять
- уменьшается глубина вложения
- предотвращаются ошибки по причине отсутствия обновления отдельных точек выхода при внесении изменений
- уменьшает объем работы компилятора для оптимизации избыточного кода ;)

```
int
fun(int a)
{
  int result = 0;
  char *buffer;

  buffer = kmalloc(SIZE, GFP_KERNEL);
  if (!buffer)
    return -ENOMEM;

  if (condition1) {
    while (loop1) {
      ...
    }
    result = 1;
    goto out_free_buffer;
  }
  ...
out_free_buffer:
  kfree(buffer);
  return result;
}
```

Распространенный тип ошибок, о котором следует помнить, — `однократное использование err`, что выглядит так:

```
err:
  kfree(foo->bar);
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  kfree(foo);
  return ret;
```

Ошибка в этом коде заключается в том, что на некоторых путях выхода `foo` принимает значение NULL. Обычно это можно исправить разделением ошибки на две метки `err_free_bar:` и `err_free_foo:`:

```
err_free_bar:
 kfree(foo->bar);
err_free_foo:
 kfree(foo);
 return ret;
```

В идеале следует моделировать ошибки, чтобы проверить все пути выхода.

### Глава 8: Комментирование

Комментарии полезны, но есть и опасность чрезмерного комментирования. НИКОГДА не пытайтесь объяснить в комментарии, КАК работает ваш код: гораздо лучше написать код так, чтобы **принцип работы** был очевиден, а объяснять плохо написанный код — это пустая трата времени.

Как правило, желательно, чтобы комментарии поясняли, ЧТО делает ваш код, а не КАК. Кроме того, постарайтесь не размещать комментарии внутри тела функции: если функция настолько сложна, что нужно отдельно комментировать ее части, скорее всего, вам надо вернуться к главе 6. Можно давать небольшие комментарии, чтобы отметить что-то особенно умное (или уродливое) или предупредить об этом, но старайтесь избегать лишнего. Вместо этого поставьте комментарии во главе функции, сообщите людям, что она делает, и, возможно, ПОЧЕМУ она это делает.

При комментировании функций Tarantool C API используйте систему комментирования Doxygen (разновидность Javadoc): то есть `@tag`, а не \\tag. Основные используемые теги: `@param`, `@retval`, `@return`, `@see`, `@note` и `@todo`.

Каждая функция, за исключением, пожалуй, очень короткой и очевидной, должна быть прокомментирована. Пример комментария функции может выглядеть следующим образом:

```
/**
 * Запись всех данных в дескриптор.
 *
 * Эта функция аналогична 'write' во всём кроме того, что она обеспечивает
 * запись всех данных в файл, если не возникает ошибка,
 * которую нельзя игнорировать.
 *
 * @retval 0  Выполнено
 * @retval 1 Ошибка (не EINTR)
 */
static int
write_all(int fd, void *data, size_t len);
```

Также важно комментировать типы данных независимо от того, базовые это типы или производные. Для этого используйте только одно объявление данных в строке (без запятой для объявления массива данных). Это оставляет вам место для небольшого комментария к каждому пункту с объяснением его использования.

Доступные структуры и важные элементы структуры также должны быть прокомментированы.

В C комментарии внутри и снаружи функции должны отличаться тем, как они начинаются. Все остальное — неправильно. Ниже приведены правильные примеры. /** используется для комментирования документации, /* — для локальных незадокументированных комментариев. Однако разница уже неявная, поэтому правило простое: снаружи функции используйте /**, внутри — /*.

```
/**
 * Комментарий снаружи функции, вариант 1.
 */

/** Комментарий снаружи функции, вариант 2. */

int
function()
{
    /* Комментарий внутри функции, вариант 1. */

    /*
     * Комментарий внутри функции, вариант 2.
     */
}
```

Если объявление функции и ее реализация разделены, то комментарий к функции должен относиться к части объявления функции. Обычно в файле заголовка. Не дублируйте комментарий.

Комментарий и сигнатура функции должны быть согласованы. Перепроверьте, что имена параметров те же, что и в комментарии, и означают одно и то же. Особенно, если вы изменили один из них, — убедитесь, что вы изменили и другой.

### Глава 9: Макросы, перечисления и уровни регистровых передач (RTL)

Имена макросов, определяющих постоянные и метки в перечислениях, пишутся заглавными буквами.

```
#define CONSTANT 0x12345
```

Рекомендуется использовать перечисления при определении нескольких связанных постоянных.

Ценятся имена макросов, написанные ЗАГЛАВНЫМИ буквами, но похожие на функции макросы можно называть, используя буквы в нижнем регистре.

Как правило, рекомендуется использовать встроенные функции для макросов, похожих на функции.

Макросы с несколькими операторами должны быть заключены в блок do - while:

```
#define macrofun(a, b, c)       \
  do {                          \
    if (a == 5)                 \
      do_this(b, c);            \
  } while (0)
```

Во время использования макросов постарайтесь избегать следующего:

1) Макросы, которые влияют на поток управления:

```
#define FOO(x)                  \
  do {                          \
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    if (blah(x) < 0)            \
      return -EBUGGERED;        \
  } while (0)
```

это **очень** плохая идея. Он выглядит как вызов функции, но выходит из `вызывающей` функции; не ломайте внутреннего анализатора у того, кто будет читать код.

2) Макросы, которые зависят от наличия локальной переменной с магическим именем:

```
#define FOO(val) bar(index, val)
```

могут показаться хорошей идеей, но они сбивают с толку, когда читаешь код, и такой код склонен ломаться от, казалось бы, невинных изменений.

3) Макросы с аргументами, которые используются как l-значения: `FOO(x) = y;`. Это вам аукнется, если кто-то, например, сделает FOO встроенной функцией.

4) Потеря приоритета: макросы, определяющие постоянные с использованием выражений, должны заключать выражение в круглые скобки. Остерегайтесь аналогичных проблем с макросами с использованием параметров.

```
#define CONSTANT 0x4000
#define CONSTEXP (CONSTANT | 3)
```

5) Конфликты в пространствах имен при определении локальных переменных в макросах, напоминающих функции:

```
#define FOO(x)              \
({                          \
  typeof(x) ret;            \
  ret = calc_ret(x);        \
  (ret);                    \
})
```

ret — обычное имя для локальной переменной; имя `__foo_ret` вряд ли вызовет конфликт с уже существующей переменной.

### Глава 10: Выделение памяти

Лучше использовать специализированные генераторы, такие как `region`, `mempool`, `smalloc`, вместо `malloc()/free()``для любых операций выделения памяти большого объема. Многократное использование ``malloc()/free()` может привести к фрагментации памяти, чего следует избегать.

Всегда освобождайте всю выделенную память, даже выделенную при запуске. Мы стремимся к тому, чтобы valgrind не находил утечек памяти, и в большинстве случаев так же легко освободить выделенную память по `free()`, как и записать подавление valgrind. Освобождение всей выделенной памяти также помогает динамическому балансированию нагрузки: предполагается, что подключаемый модуль может динамически загружаться и выгружаться несколько раз, перезагрузка не должна приводить к утечке памяти.

### Глава 11: Болезнь встраивания (inline)

Похоже, что распространено ошибочное представление о том, что в gcc есть волшебная опция ускорения, называемая встраиванием `inline`. Хотя использование встроенных строк может быть оправдано, довольно часто это не так. Избыток ключевого слова inline приводит к увеличению ядра, что в свою очередь, замедляет работу системы в целом из-за большего объема отпечатка icache для процессора и просто потому, что для pagecache доступно меньше памяти. Просто подумайте: непопадание в pagecache вызывает поиск по диску, который легко занимает 5 миллисекунд. Есть МНОГО циклов процессора, которые могут пройти в эти 5 миллисекунд.

Общее правило состоит в том, чтобы не вводить встраивание в функции, содержащие больше трех строк кода. Исключением из этого правила являются случаи, когда параметр известен как постоянная времени компиляции, и в результате вы *знаете*, что компилятор сможет оптимизировать большую часть ваших функций во время компиляции.

Часто утверждают, что беспроигрышным вариантом будет встраивание статических функций, используемых только один раз, поскольку нет компромиссов пространства. Хотя это технически правильно, gcc способен автоматически встраивать их, а проблема удаления встроенного, если появляется второй пользователь, перевешивает потенциальную ценность подсказки для gcc делать что-то, что он сделал бы в любом случае.

### Глава 12: Возвращаемые значения и имена функций

Функции могут возвращать значения множества различных типов, и одним из наиболее распространенных является значение, которое указывает, была функция выполнена или нет.

В 99.99999% случаев в Tarantool при выполнении функции возвращается 0, в случае ошибки — ненулевое значение (обычно -1). Ошибки сохраняются в рабочей области диагностики (одна на файбер). Результатом функции никогда не будет код ошибки.

Функции, возвращаемое значение которых является фактическим результатом вычисления, а не указанием того, удалось ли выполнить вычисление, не подпадают под это правило. Обычно они указывают на сбой, возвращая некое недопустимое значение. Типичными примерами будут функции, возвращающие указатели; чтобы сообщить об ошибке, они используют NULL.

### Глава 13: Редакторские строки режима (modelines) и прочий хлам

Некоторые редакторы могут интерпретировать встроенную в исходные файлы информацию о конфигурации, указанную специальными маркерами. Например, emacs интерпретирует строки, помеченные следующим образом:

```
-*- mode: c -*-
```

Или так:

```
/*
Local Variables:
compile-command: "gcc -DMAGIC_DEBUG_FLAG foo.c"
End:
*/
```

Vim интерпретирует маркеры, которые выглядят так:

```
/* vim:set sw=8 noet */
```

Не включайте их в исходные файлы. У людей есть свои собственные настройки редакторов, и ваши исходные файлы не должны их переопределять. Это относится к маркерам для отступов и конфигурации режима. У других людей могут быть свои собственные режимы или другие волшебные методы для правильной работы отступов.

### Глава 14: Условная компиляция

По возможности не используйте препроцессорные директивы (`#if`, `#ifdef`) в файлах .c. Это затрудняет чтение кода и понимание логики. Вместо этого используйте такие директивы в файле заголовка, чтобы определить функции, используемые в этих файлах .c с заглушками в виде холостых команд в случае #else, а затем вызывайте эти функции безусловно из файлов .c. Компилятор не будет генерировать код для вызовов заглушек, при этом результат останется таким же, но логику будет проще понять.

Лучше компилировать целые функции, а не части функций или части выражений. Вместо того, чтобы вставить `#ifdef` в выражение, выделите часть или все выражение в отдельную вспомогательную функцию и примените условие к этой функции.

Если у вас есть функция или переменная, которая может не использоваться в конкретной конфигурации, и компилятор предупредит о том, что она использоваться не будет, не компилируйте ее и используйте для этого #if.

В конце любого крупного блока `#if` или `#ifdef` (более нескольких строк) после #endif в той же строке поместите комментарий, отмечающий используемое условное выражение. Например:

```
#ifdef CONFIG_SOMETHING
...
#endif /* CONFIG_SOMETHING */
```

### Глава 15: Файлы заголовка

В заголовках используйте `#pragma once`. Для защиты заголовков мы используем такую конструкцию:

```
#ifndef THE_HEADER_IS_INCLUDED
#define THE_HEADER_IS_INCLUDED

// ... код заголовка ...

#endif // THE_HEADER_IS_INCLUDED
```

Работает нормально, но имя защиты `THE_HEADER_IS_INCLUDED` обычно перестает действовать при перемещении или переименовании файла. Это особенно неудобно, если у нескольких файлов одинаковое имя в проекте, но разные пути. Например, у нас есть 3 файла `error.h`, а это значит, что для каждого из них нужно придумать новое имя защиты заголовка, и не забыть обновить их при перемещении или переименовании файлов.

По этой причине мы и используем `#pragma once` во всем новом коде, что сокращает файл заголовка до такого:

```
#pragma once

// ... код заголовка ...
```

#### Глава 16: Прочие положения

- Мы не применяем оператор `!` к значениям, отличным от boolean. То есть, чтобы проверить, не равно ли целое число 0, вы используете `!= 0`. Чтобы проверить, что указатель не NULL, используете `!= NULL`. То же самое для `==`.
- Допускаются расширения GNU C99. Можно смешивать операторы и объявления в выражениях.
- Не слишком актуальный список всех расширений семейства языка C можно найти по ссылке: http://gcc.gnu.org/onlinedocs/gcc-4.3.5/gcc/C-Extensions.html

#### Приложение I: Источники

- Керниган Брайан В., Ричи Деннис М. Язык программирования Си. Prentice Hall, Inc., 1988. ISBN 0-13-110362-8 (в мягкой обложке), 0-13-110370-9 (в твердом переплете).
- Керниган Брайан В., Пайк Роб. Практика программирования. Addison-Wesley, Inc., 1999. ISBN 0-201-61586-X.
- Рекомендации GNU в соответствии с K&R и данным текстом – для **cpp**, **gcc**, **gcc internals** и **indent**
- Рабочая группа по международной стандартизации языка программирования C WG14
- Стиль программирования ядра, автор greg@kroah.com, презентация на OLS 2002

### 10.5.4 Руководство по написанию кода на Python

#### Введение

Данный документ описывает соглашение о том, как писать код для языка Python, включая стандартную библиотеку, входящую в состав Python. Посмотрите также на сопутствующую PEP (Python enhanced proposal – заявку на улучшение языка Python), описывающую, какого стиля следует придерживаться при написании кода на C в реализации языка Python[1].

Данный документ, а также PEP 257 (Документирование кода) созданы на основе оригинала рекомендаций Гуидо ван Россума с добавлениями от Барри[2].

#### A Foolish Consistency is the Hobgoblin of Little Minds («Безрассудная согласованность сбивает с толку мелкие умы»)

Одна из ключевых идей Гвидо заключается в том, что код читается намного чаще, чем пишется. И рекомендации по стилю программирования предназначены улучшить читаемость кода и сделать его согласованным во множестве проектов на языке Python. Как написано в PEP 20, «Читаемость имеет значение».

В руководстве речь идет о согласованности. Согласованность с руководством очень важна. Согласованность внутри проекта еще важнее. А согласованность в пределах модуля или функции – самое важное.

Но очень важно понимать, когда можно отойти от рекомендаций, потому что руководство неприменимо. Если вы сомневаетесь, используйте свой опыт. Просто посмотрите на другие примеры и решите, какой выглядит лучше. И не бойтесь спросить!

[1] ван Россум Гвидо. PEP 7, Руководство по программированию на языке C
[2] Руководство Барри по GNU Mailman

Правила можно нарушить по одной из этих причин:

1. Если применение правила сделает код менее читаемым даже для того, кто привык читать код, написанный по правилам.
2. Чтобы не отступать по стилю от уже написанного не по правилам кода (возможно, в силу исторических причин) – впрочем, это может быть возможность причесать чужой код (в стиле XP).

### Размещение кода

#### Отступы

Используйте 4 пробела на каждый уровень отступа.

Если вы не хотите наводить путаницу в очень старом коде, можете продолжать использовать отступы в 8 пробелов.

Продолжения строк должны выравнивать переносимые элементы либо вертикально, используя подразумевающееся объединение строк в скобках (круглых, квадратных или фигурных), либо с использованием висячего отступа. При использовании висячего отступа необходимо применять следующие соображения: на первой строке не должно быть аргументов, а остальные строки должны четко восприниматься как продолжение строки.

Правильно:

```
# выравнивание по открывающему разделителю
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# больше отступов, чтобы данный сегмент отличался от остальных.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

Неправильно:

```
# запрещены аргументы на первой строке, если не используется вертикальное выравнивание
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# необходимы дополнительные отступы для четких отличий
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Возможно:

```
# Нет необходимости в дополнительных отступах.
foo = long_function_name(
  var_one, var_two,
  var_three, var_four)
```

Закрывающие круглые/квадратные/фигурные скобки в многострочных конструкциях могут находиться либо под первым символом последней строки списка (не пробелом), например:

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
    )
```

либо под первым символом строки, с которой начинается многострочная конструкция:

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
)
```

### Табуляция или пробелы?

Никогда не смешивайте символы табуляции и пробелы.

Самый распространенный способ отступов в Python – пробелы. На втором месте – отступы только с использованием табуляции. Код, в котором используются и те, и другие типы отступов, следует исправить так, чтобы отступы в нем были расставлены только с помощью пробелов. При вызове интерпретатора в командной строке с параметром `-t` он выдаст предупреждение в случае использовании смешанного стиля в отступах. Запустив интерпретатор с параметром -tt, вы получите в этих местах ошибки. Рекомендуем использовать эти опции!

В новых проектах для отступов настоятельно рекомендуется использовать только пробелы. Во многих редакторах можно легко это делать.

### Максимальная длина строки

Ограничьте максимальную длину строки 79 символами.

Пока еще есть немало устройств, где длина строки ограничена 80 символами; к тому же, ограничив ширину окна 80 символами, мы можем расположить несколько окон рядом друг с другом. Автоматический перенос строк на таких устройствах нарушит форматирование, и код будет труднее понять. Поэтому ограничьте длину строки 79 символами. Для длинных блоков текста (строки документации или комментарии) рекомендуется ограничиваться 72 символами.

Предпочтительный способ переноса длинных строк – использование подразумевающегося продолжения строки между обычными, квадратными и фигурными скобками. Длинные строки можно разбить на несколько строк в скобках. Это лучше, чем использовать обратную косую черту для продолжения строки.

Обратную косую черту можно использовать время от времени. Например, длинный оператор with не может работать с неявными продолжениями, так что обратная косая черта здесь подойдет:

```
with open('/path/to/some/file/you/want/to/read') as file_1, \
        open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
```

Еще один такой случай – операторы `assert`.

Делайте правильные отступы для перенесенной строки. Предпочтительнее вставить перенос строки *после* логического оператора, а не перед ним. Например:

```
class Rectangle(Blob):

    def __init__(self, width, height,
                 color='black', emphasis=None, highlight=0):
        if (width == 0 and height == 0 and
            color == 'red' and emphasis == 'strong' or
            highlight > 100):
            raise ValueError("sorry, you lose")
        if width == 0 and height == 0 and (color == 'red' or
                                           emphasis is None):
            raise ValueError("I don't think so -- values are %s, %s" %
                             (width, height))
        Blob.__init__(self, width, height,
                      color, emphasis, highlight)
```

### Пустые строки

Отделяйте функции верхнего уровня и определения классов двумя пустыми строками.

Определения методов в пределах класса отделяйте одной пустой строкой.

Также можно добавлять пустые строки (не слишком часто) для выделения групп связанных функций. Пустые строки не стоит добавлять между несколькими связанными программами в одну строку (например, в формальной реализации).

Не слишком часто можно добавлять пустые строки в коде функций, чтобы отделить друг от друга логические части.

Python расценивает символ control+L (или ^L) как пробел. Многие редакторы обрабатывают его как разрыв страницы, поэтому его можно использовать для выделения логических части в файле на разных страницах. Обратите внимание, что не все редакторы распознают control+L и могут на его месте отображать другой символ.

### Кодировка (PEP 263)

В коде ядра Python всегда должна использоваться кодировка ASCII или Latin-1 (также известную как ISO-8859-1). Начиная с версии Python 3.0, предпочтительной является кодировка UTF-8, а не Latin-1 (см. PEP 3120).

Для файлов с ASCII не следует объявлять кодировку. Используйте Latin-1 (или UTF-8), только если необходимо указать в комментарии или строке документации имя автора, содержащее в себе символ из Latin-1. В остальных случаях рекомендуется использовать управляющие символы x, u или U, чтобы вставить в строку символы не из ASCII.

Начиная с версии Python 3.0 и выше, в стандартной библиотеке действует следующая политика (см. PEP 3131): все идентификаторы в стандартной библиотеке Python ДОЛЖНЫ содержать только

ASCII-символы и означать английские слова везде, где это возможно (во многих случаях используются сокращения или неанглийские технические термины). Кроме того, строки и комментарии также должны содержать лишь ASCII-символы. Исключения составляют: (a) тестовые сценарии для тестирования функций программы в других кодировках, и (b) имена авторов. Авторы, в именах которых есть буквы не из латинского алфавита, должны транслитерировать свои имена в латиницу.

В проектах с открытым кодом для широкой аудитории также рекомендуется использовать это правило.

### Импорт

- Импорт разных модулей должен быть на разных строках, например:

```
Yes: import os
     import sys

No:  import sys, os
```

  В то же время, можно писать вот так:

```
from subprocess import Popen, PIPE
```

- Импорт всегда нужно делать в начале файла сразу после комментариев к модулю и строк документации, перед объявлением глобальных переменных и постоянных.

  Группируйте импорты в следующем порядке:

  1. импорты стандартной библиотеки
  2. импорты сторонних библиотек
  3. импорты модулей текущего проекта

  Между группами импортов вставляйте пустую строку.

  Указывайте все необходимые спецификации __all__ после импортов.

- Относительные импорты крайне не рекомендуются. Всегда указывайте абсолютный путь к модулю для всех видов импорта. Даже сейчас, когда PEP 328 реализован в версии Python 2.5, явно использовать относительные импорты не рекомендуется. Абсолютные импорты более независимы и, как правило, обладают лучшей читаемостью.

- При импорте класса из модуля с классами, обычно можно писать так:

```
from myclass import MyClass
from foo.bar.yourclass import YourClass
```

  Если такое написание вызывает конфликт локальных имен, пишите:

```
import myclass
import foo.bar.yourclass
```

  И используйте «myclass.MyClass» и «foo.bar.yourclass.YourClass».

### Пробелы в выражениях и операторах

#### Наболевшие вопросы

Избегайте использования пробелов в следующих ситуациях:

- Перед круглыми, фигурными и квадратными скобками и после них:

```
Yes: spam(ham[1], {eggs: 2})
No:  spam( ham[ 1 ], { eggs: 2 } )
```

- Сразу перед запятой, точкой с запятой, двоеточием:

```
Yes: if x == 4: print x, y; x, y = y, x
No:  if x == 4 : print x , y ; x , y = y , x
```

- Сразу перед открывающей скобкой, после которой начинается список аргументов при вызове функции:

```
Yes: spam(1)
No:  spam (1)
```

- Сразу перед открывающей скобкой, после которой идет индекс или срез:

```
Yes: dict['key'] = list[index]
No:  dict ['key'] = list [index]
```

- Больше одного пробела вокруг оператора присваивания (или другого) для того, чтобы выровнять его с другим оператором:

  Правильно:

```
x = 1
y = 2
long_variable = 3
```

  Неправильно:

```
x             = 1
y             = 2
long_variable = 3
```

### Прочие рекомендации

- Всегда окружайте эти знаки двухместных операций пробелами по одному с каждой стороны: присваивание (=), комбинированное присваивание (+=, -= и т.д.), сравнения (==, <, >, !=, <>, <=, >=, in, not in, is, is not), логические операторы (and, or, not).
- Если используются знаки операций с разными приоритетами, рассмотрите возможность добавить пробелы вокруг операций с самым низким приоритетом. Судите сами, однако, никогда не используйте больше одного пробела, и всегда используйте одинаковое количество пробелов по обе стороны от знака.

  Правильно:

```
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

Неправильно:

```
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

- Не используйте пробелы для отделения знака =, когда он употребляется для обозначения аргумента ключевого слова или значения параметра по умолчанию.

  Правильно:

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

  Неправильно:

```
def complex(real, imag = 0.0):
    return magic(r = real, i = imag)
```

- Не рекомендуется использовать составные операторы (несколько операторов в одной строке).

  Правильно:

```
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

  Скорее неправильно:

```
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
```

- Иногда можно разместить тело цикла if/for/while в той же строке, но если операторов несколько, никогда так не делайте. И избегайте свертывания таких длинных строк!

  Скорее неправильно:

```
if foo == 'blah': do_blah_thing()
for x in lst: total += x
while t < 10: t = delay()
```

  Точно неправильно:

```
if foo == 'blah': do_blah_thing()
else: do_non_blah_thing()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
try: something()
finally: cleanup()

do_one(); do_two(); do_three(long, argument,
                             list, like, this)

if foo == 'blah': one(); two(); three()
```

### Комментарии

Комментарии, которые противоречат коду, хуже, чем отсутствие комментариев. Всегда считайте первоочередной задачей исправить комментарии, если меняется код!

Комментарии должны представлять собой законченные предложения. Если комментарием будет фраза или предложение, первое слово должно быть написано с заглавной буквы, если только это не идентификатор, который пишется со строчной буквы (никогда не меняйте регистр идентификаторов!).

Если комментарий короткий, точку в конце предложения можно опустить. Блок комментариев обычно состоит из одного или более абзацев, составленных из полных предложений, поэтому каждое предложение должно заканчиваться точкой.

После точки в конце предложения следует ставить два пробела.

Если вы пишете на английском языке, не забывайте о рекомендациях Странка и Уайта по стилю.

Разработчики на языке Python из неанглоязычных стран, пишите комментарии на английском, если только вы не уверены на 120%, что ваш код никогда не будут читать люди, не знающие вашего родного языка.

### Блок комментариев

Блок комментариев обычно сопровождает фрагмент кода (или весь код), который за ним следует, и находится на том же уровне отступов, что и сам код. Каждая строка блока комментариев должна начинаться с символа # и одного пробела после него (если только в самом тексте комментария нет отступов).

Абзацы в пределах блока комментариев отделяются строкой, состоящей из одного символа #.

### Комментарии в строке с кодом

Старайтесь реже использовать подобные комментарии.

Встроенный комментарий находится в той же строке, что и оператор. Такие комментарии должны отделяться от оператора хотя бы двумя пробелами. Они должны начинаться с символа # и одного пробела.

Комментарии в строке с кодом не нужны и в действительности отвлекают от чтения, если они объясняют очевидное. Не пишите так:

```
x = x + 1                 # Увеличение x
```

Иногда, впрочем, они полезны:

```
x = x + 1                 # Место для рамки окна
```

### Строки документации

Соглашения о написании хорошей документации (docstrings) увековечены в PEP 257.

- Пишите документацию для всех доступных модулей, функций, классов, методов. Строки документации необязательны для внутренних методов, но нужно добавить комментарий о том, что делает метод. Комментарий должен идти после строки `def`.
- PEP 257 объясняет, как правильно и хорошо писать документацию. Следует отметить, что очень важно, чтобы закрывающие """ стояли на отдельной строке, а предпочтительно, чтобы перед ними была и пустая строка, например:

```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.

"""
```

- Для однострочной документации можно оставить закрывающие """ на той же строке.

### Контроль версий

Если вам нужно использовать Subversion, CVS или RCS в ваших исходных кодах, делайте это следующим образом:

```
__version__ = "$Revision$"
# $Source$
```

Эти строки следует указывать после документации модуля перед любым другим кодом, отделяя их пустыми строками сверху и снизу.

### Соглашения по именованию

Соглашения по именованию переменных в Python довольно запущены, поэтому полной согласованности невозможно будет добиться. Тем не менее, ниже мы приводим список рекомендованных стандартов именования. Новые модули и пакеты (включая сторонние) должны быть написаны в соответствии с этими стандартами, но если уже существующая библиотека написана в другом стиле, предпочтительно поддерживать согласованность.

### Описание: Стили имен

Существует много различных стилей именования. Полезно распознавать, какой стиль именования используется независимо от того, для чего он используется.

Обычно различают следующие стили именования:

- `b` (отдельная строчная буква)
- `B` (отдельная заглавная буква)
- `lowercase` (слово в нижнем регистре)

- `lower_case_with_underscores` (слова из строчных букв с символами подчеркивания)
- `UPPERCASE` (заглавные буквы)
- UPPERCASE_WITH_UNDERSCORES (слова из заглавных букв с символами подчеркивания)
- `CapitalizedWords` (слова с заглавными буквами, или CapWords, или CamelCase – называется так, потому что прописные буквы внутри слова напоминают горбы верблюда[3]). Иногда называется StudlyCaps.

  Примечание: когда вы используете аббревиатуры в стиле CapWords, пишите все буквы аббревиатуры заглавными. HTTPServerError выглядит лучше, чем HttpServerError.
- mixedCase (отличается от CapitalizedWords тем, что первое слово начинается со строчной буквы!)
- Capitalized_Words_With_Underscores (слова с заглавными буквами и символами подчеркивания – уродливо!)

Еще есть стиль, в котором к именам из одной логической группы добавляется короткий уникальный префикс. Этот стиль редко используется в Python, но упомянем его для полноты изложения. Например, функция `os.stat()` возвращает кортеж, имена в котором традиционно выглядят так: `st_mode`, `st_size`, `st_mtime` и так далее. (Так сделано, чтобы подчеркнуть соответствие этих полей структуре системных вызовов POSIX, что помогает знакомым с ней разработчикам).

В библиотеке X11 используется префикс X для всех доступных функций. В Python этот стиль считается лишним, потому что перед полями и именами методов стоит имя объекта, а перед именами функций стоит имя модуля.

Кроме того, используются следующие специальные формы записи имен с добавлением символа подчеркивания в начало или конец имени (их можно использовать с любым типом регистра):

- `_single_leading_underscore`: слабый индикатор «для внутреннего пользования». Например, `from M import *` не будет импортировать объекты, имена которых начинаются с символа подчеркивания.
- `single_trailing_underscore_`: используется по соглашению во избежание конфликтов с ключевыми словами Python, например:

```
Tkinter.Toplevel(master, class_='ClassName')
```

- `__double_leading_underscore`: изменяет имя атрибута класса (в классе FooBar, `__boo` становится `_FooBar__boo`; см. ниже).
- `__double_leading_and_trailing_underscore__`: «волшебные» объекты или атрибуты, которые находятся в live in в пространствах имен, управляемых пользователем. Например, `__init__`, `__import__` или `__file__`. Не придумывайте такие имена, используйте их только так, как написано в документации.

### Предписания: соглашения по именованию

#### Имена, которых следует избегать

Никогда не используйте символы „l“ (строчная латинская буква эль), „O“ (заглавная латинская буква о) или „I“ (заглавная латинская буква ай) в качестве однобуквенных имен переменных.

В некоторых шрифтах эти символы неотличимы от цифр один и ноль. Если нельзя обойтись без „l“, пишите вместо нее „L“.

[3] Страница Википедии о CamelCase

### Имена модулей и пакетов

Имена модулей должны быть короткими и состоять из строчных букв. Можно использовать и символы подчеркивания, если это улучшает читаемость. Имена пакетов Python также должны быть короткими и состоять из строчных букв, но здесь символы подчеркивания не приветствуются.

Так как имена модулей отображаются в именах файлов, а некоторые файловые системы являются нечувствительными к регистру символов и обрезают длинные имена, очень важно использовать достаточно короткие имена модулей – это не проблема в Unix, но может стать проблемой при переносе кода в старые версии Windows, Mac или DOS.

Если для модуля расширения, написанного на C или C++, есть сопутствующий Python-модуль, содержащий интерфейс более высокого уровня (например, более объектно-ориентированный), модуль C/C++ начинается с символа подчеркивания (например, `_socket`).

### Имена классов

Все имена классов должны соответствовать CapWords почти без исключений. Классы для внутреннего использования могут также начинаться с символа подчеркивания.

### Имена исключений

Так как исключения должны быть классами, к исключениям применяются правила именования классов. Однако вы можете добавить суффикс «Error» в конце имени (если исключение действительно является ошибкой).

### Имена глобальных переменных

(Будем надеяться, что такие имена используются только в пределах одного модуля.) Применяются те же правила, что и для имен функций.

В модули, которые предназначены для использования с помощью `from M import *`, следует добавить механизм `__all__`, чтобы предотвратить экспорт глобальных переменных, или же использовать старое соглашение, добавляя перед именами таких глобальных переменных один символ подчеркивания (которым можно обозначить глобальные переменные, которые используются только внутри модуля).

### Имена функций

Имена функций должны состоять из строчных букв, а слова разделяться символами подчеркивания, чтобы улучшить читаемость.

mixedCase допускается только в тех местах, где уже преобладает такой стиль (например, threading.py), для обратной совместимости.

### Аргументы функций и методов

Всегда используйте `self` в качестве первого аргумента метода экземпляра.

Всегда используйте `cls` в качестве первого аргумента метода класса.

Если имя аргумента функции конфликтует с зарезервированным ключевым словом, обычно лучше добавить в конец имени символ подчеркивания, а не сокращать слово или искажать его. Таким образом, `class_` лучше, чем `clss`. (Возможно, будет лучше избегать конфликта имен путем подбора синонима).

### Имена методов и переменные экземпляров

Используйте тот же стиль, что и для имен функций: они должны состоять из строчных букв, а слова разделяться символами подчеркивания, чтобы улучшить читаемость.

Используйте только один символ подчеркивания в начале слова для внутренних методов и переменных экземпляров.

Чтобы избежать конфликта имен с подклассами, добавьте два символа подчеркивания в начале слова, чтобы включить механизм изменения имен в Python.

Python изменяет эти имена: если в классе Foo есть атрибут с именем `__a`, к нему нельзя обратиться через `Foo.__a`. (Настойчивый пользователь всё равно может получить доступ через `Foo._Foo__a`.) Вообще, двойное подчеркивание в начале имени должно использоваться только во избежание конфликта имен с атрибутами классов, предназначенных для разделения на подклассы.

Примечание: есть некоторые разногласия по поводу использования имен __names (см. ниже).

### Постоянные

Постоянные обычно объявляются на уровне модуля и записываются только заглавными буквами, а слова разделяются символами подчеркивания. Например: MAX_OVERFLOW, TOTAL.

### Проектирование наследования

Обязательно решите, каким должен быть метод класса или переменная экземпляра класса (в общем, атрибут) – доступными (public) или внутренними (non-public). Если вы сомневаетесь, делайте их внутренними. Потом будет проще открыть к ним доступ, чем наоборот.

Доступные атрибуты – это такие атрибуты, которые будут использовать потребители ваших классов, и вы должны быть уверены в обратной совместимости. Внутренние атрибуты, в свою очередь, не предназначены для использования третьими лицами, поэтому вы можете не гарантировать, что не измените или не удалите эти атрибуты.

Мы не используем термин «закрытый» (private), потому что на самом деле в Python таких атрибутов не бывает (без ненужных дополнительных усилий).

Другой тип атрибутов классов принадлежит так называемому API подклассов (в других языках они часто называются защищенными – «protected»). Некоторые классы предназначены для наследования другими классами, которые расширяют или изменяют поведение базового класса. Когда вы проектируете такой класс, решите и явным образом укажите, какие атрибуты являются доступными (public), какие относятся к API подклассов (subclass API), а какие используются только базовым классом.

С учетом вышесказанного, сформулируем рекомендации:

- В начале имени доступных атрибутов не должно быть символов подчеркивания.

- Если имя доступного атрибута конфликтует с ключевым словом языка, добавьте в конец имени один символ подчеркивания. Это более предпочтительно, чем сокращать слово или искажать его (однако, у этого правила есть исключение: „cls“ – это предпочтительное написание любой переменной или аргумента, который означает класс, а особенно первого аргумента метода класса).

  **Примечание 1:** См. рекомендации по именам аргументов выше для методов класса.

- Назовите простые открытые атрибуты понятными именами и не пишите сложные методы доступа и изменения (accessor/mutator). Следует помнить, что в Python очень легко расширить поведение функции, если потребуется. В этом случае используйте свойства (properties), чтобы скрыть функциональную реализацию за синтаксисом доступа к атрибутам.

  **Примечание 1:** Свойства работают только в классах нового стиля (new-style classes).

  **Примечание 2:** Постарайтесь избавиться от побочных эффектов, связанных с функциональным поведением, хотя такие вещи, как кэширование, вполне допустимы.

  **Примечание 3:** Избегайте использовать вычислительно затратные операции, потому что из-за записи с помощью атрибутов создается впечатление, что доступ происходит (относительно) быстро.

- Если ваш класс предназначен для разделения на подклассы, но некоторые атрибуты не должны наследоваться подклассами, подумайте о добавлении в имена двух символов подчеркивания в начале и ни одного в конце. Механизм изменения имен в Python сработает так, что имя класса добавится к имени такого атрибута. Это позволит избежать конфликта имен, если в подклассах случайно появятся атрибуты с такими же именами.

  **Примечание 1:** Обратите внимание, что только имена простых классов используются в измененном имени, поэтому если в подклассе будет то же имя класса и имя атрибута, то снова возникнет конфликт имен.

  **Примечание 2:** Механизм изменения имен может затруднить отладку или работу с `__getattr__()`. Тем не менее, алгоритм хорошо документирован и легко реализуется вручную.

  **Примечание 3:** Не всем нравится механизм изменения имен. Постарайтесь достичь компромисса между необходимостью избежать конфликта имен и возможностью доступа к этим атрибутам.

### Использованная литература

### Защита авторских прав

Автор:

- Гвидо ван Россум <guido@python.org>
- Барри Ворсо <barry@python.org>

### 10.5.5 Lua style guide

Для вдохновения:

- https://github.com/Olivine-Labs/lua-style-guide
- http://dev.minetest.net/Lua_code_style_guidelines
- http://sputnik.freewisdom.org/en/Coding_Standard

Programming style is art. There is some arbitrariness to the rules, but there are sound rationales for them. It is useful not only to provide sound advice on style but to understand the underlying rationale behind the style recommendations:

- http://mindprod.com/jgloss/unmain.html
- http://www.oreilly.com/catalog/perlbp/
- http://books.google.com/books?id=QnghAQAAIAAJ

The Zen of Python is good. Understand it and use wisely:

> Красивое лучше, чем уродливое.
> Явное лучше, чем неявное.
> Простое лучше, чем сложное.
> Сложное лучше, чем запутанное.
> Плоское лучше, чем вложенное.
> Разреженное лучше, чем плотное.
> Читаемость имеет значение.
> Особые случаи не настолько особые, чтобы нарушать правила.
> При этом практичность важнее безупречности.
> Ошибки никогда не должны замалчиваться.
> Если не замалчиваются явно.
> Встретив двусмысленность, отбрось искушение угадать.
> Должен существовать один – и, желательно, только один – очевидный способ сделать это.
> Хотя он поначалу может быть и не очевиден.
> Сейчас лучше, чем никогда.
> Хотя никогда зачастую лучше, чем прямо сейчас.
> Если реализацию сложно объяснить – идея плоха.
> Если реализацию легко объяснить – идея, возможно, хороша.
> Пространства имен – отличная штука! Сделаем побольше!
>
> https://www.python.org/dev/peps/pep-0020/

#### Indentation and formatting

- 4 spaces instead of tabs. PIL suggests using two spaces, but a programmer looks at code from 4 to 8 hours a day, so it's simpler to distinguish indentation with 4 spaces. Why spaces? Similar representation everywhere.

  Можно использовать строки режима (modelines) vim:

```
-- vim:ts=4 ss=4 sw=4 expandtab
```

- Файл должен заканчиваться на один символ переноса строки, но не должен заканчиваться на пустой строке (два символа переноса строки).
- Отступы всех do/while/for/if/function должны составлять 4 пробела.
- Related `or`/`and` in `if` must be enclosed in the round brackets (). Example:

```
-- Good
if (a == true and b == false) or (a == false and b == true) then
    <...>
end

-- Bad
if a == true and b == false or a == false and b == true then
    <...>
end

-- Good but not explicit
if a ^ b == true then
end
```

- Преобразование типов

  Не используйте конкатенацию для конвертации в строку или в число (вместо этого воспользуйтесь `tostring`/`tonumber`):

```
-- Bad
local a = 123
a = a .. ''

-- Good
local a = 123
a = tostring(a)

-- Bad
local a = '123'
a = a + 5 -- 128

-- Good
local a = '123'
a = tonumber(a) + 5 -- 128
```

- Постарайтесь избегать несколько вложенных `if` с общим телом оператора:

```
-- Good
if (a == true and b == false) or (a == false and b == true) then
    do_something()
end

-- Bad
if a == true then
    if b == false then
        do_something()
    end
if b == true then
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    if a == false then
        do_something()
    end
end
```

- Избегайте множества конкатенаций в одном операторе, лучше использовать `string.format`:

```
-- Bad
function say_greeting(period, name)
    local a = "good  " .. period .. ", " .. name
end

-- Good
function say_greeting(period, name)
    local a = string.format("good %s, %s", period, name)
end

-- Best
local say_greeting_fmt = "good %s, %s"
function say_greeting(period, name)
    local a = say_greeting_fmt:format(period, name)
end
```

- Используйте `and`/`or` для указания значений переменных, используемых по умолчанию,

```
-- Good
function(input)
    input = input or 'default_value'
end

-- Ok but excessive
function(input)
    if input == nil then
        input = 'default_value'
    end
end
```

- операторов `if` и возврата:

```
-- Good
if a == true then
    return do_something()
end
do_other_thing()

-- Bad
if a == true then
    return do_something()
else
    do_other_thing()
end
```

- Использование пробелов:

– Don't use spaces between function name and opening round bracket. Split arguments with one whitespace character:

```
-- Bad
function name (arg1,arg2,...)
end

-- Good
function name(arg1, arg2, ...)
end
```

– Add a space after comment markers:

```
while true do -- Inline comment
    -- Comment
    do_something()
end
--[[
Multiline
comment
]]--
```

– Surrounding operators:

```
-- Bad
local thing=1
thing = thing-1
thing = thing*1
thing = 'string'..'s'

-- Good
local thing = 1
thing = thing - 1
thing = thing * 1
thing = 'string' .. 's'
```

– Add a space after commas in tables:

```
-- Bad
local thing = {1,2,3}
thing = {1 , 2 , 3}
thing = {1 ,2 ,3}

-- Good
local thing = {1, 2, 3}
```

– Add a space in map definitions after equals signs and commas:

```
-- Bad
return {1,2,3,4}
return {
    key1 = val1,key2=val2
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Good
return {1, 2, 3, 4}
return {
    key1 = val1, key2 = val2,
    key3 = val3
}
```

You can also use alignment:

```
return {
    long_key  = 'vaaaaalue',
    key       = 'val',
    something = 'even better'
}
```

– Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between several related one-liners (for example, a set of dummy implementations).

Use blank lines in functions (sparingly) to indicate logical sections:

```
-- Bad
if thing ~= nil then
    -- ... stuff ...
end
function derp()
    -- ... stuff ...
end
local wat = 7

-- Good
if thing ~= nil then
    -- ... stuff ...
end

function derp()
    -- ... stuff ...
end

local wat = 7
```

– Delete whitespace at EOL (strongly forbidden. Use `:s/\s\+$//gc` in vim to delete them).

### Avoid global variables

Avoid using global variables. In exceptional cases, start the name of such a variable with `_G`, add a prefix, or add a table instead of a prefix:

```
-- Very bad
function bad_global_example()
end

function good_local_example()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
end
-- Good
_G.modulename_good_local_example = good_local_example

-- Better
_G.modulename = {}
_G.modulename.good_local_example = good_local_example
```

Always use a prefix to avoid name conflicts.

### Именование

- Names of variables/»objects» and «methods»/functions: snake_case.
- Names of «classes»: CamelCase.
- Private variables/methods (future properties) of objects start with underscores `<object>._<name>`. Avoid syntax like `local function private_methods(self) end`.
- Boolean: naming `is_<...>`, `isnt_<...>`, `has_`, `hasnt_` is good style.
- For «very local» variables:
  - `t` is for tables
  - `i`, `j` are for indexing
  - `n` is for counting
  - `k`, `v` is what you get out of `pairs()` (are acceptable, `_` if unused)
  - `i`, `v` is what you get out of `ipairs()` (are acceptable, `_` if unused)
  - `k`/`key` is for table keys
  - `v`/`val`/`value` is for values that are passed around
  - `x`/`y`/`z` is for generic math quantities
  - `s`/`str`/`string` is for strings
  - `c` is for 1-char strings
  - `f`/`func`/`cb` are for functions
  - `status, <rv>..` or `ok, <rv>..` is what you get out of pcall/xpcall
  - `buf, sz` is a (buffer, size) pair
  - `<name>_p` is for pointers
  - `t0`.. is for timestamps
  - `err` is for errors
- Abbreviations are acceptable if they're very common or if they're unambiguous and you've documented them.
- Global variables are spelled in ALL_CAPS. If it's a system variable, it starts with an underscore (`_G`/`_VERSION`/..).
- Modules are named in snake_case (avoid underscores and dashes): for example, „luasql", not „Lua-SQL".
- `*_mt` and `*_methods` defines metatable and methods table.

### Идиомы и шаблоны

Всегда пользуйтесь круглыми скобками при вызове функций, за исключением множественных случаев (распространенные идиомы в Lua):

- функции `*.cfg{ }` (`box.cfg`/`memcached.cfg`/..)
- функция `ffi.cdef[[ ]]`

Avoid the following constructions:

- <func>“<name>“. Strongly avoid require“..“.
- `function object:method() end`. Use `function object.method(self) end` instead.
- Semicolons as table separators. Only use commas.
- Semicolons at the end of line. Use semicolons only to split multiple statements on one line.
- Unnecessary function creation (closures/..).

Avoid implicit casting to boolean in `if` conditions like `if x then` or `if not x then`. Such expressions will likely result in troubles with *box.NULL*. Instead of those conditions, use `if x ~= nil then` and `if x == nil then`.

### Модули

Don't start modules with license/authors/descriptions, you can write it in LICENSE/AUTHORS/README files. To write modules, use one of the two patterns (don't use `modules()`):

```
local M = {}

function M.foo()
    ...
end

function M.bar()
    ...
end

return M
```

или

```
local function foo()
    ...
end

local function bar()
    ...
end

return {
    foo = foo,
    bar = bar,
}
```

### Комментирование

Don't forget to comment your Lua code. You shouldn't comment Lua syntax (assume that the reader already knows the Lua language). Instead, tell about functions/variable names/etc.

Start a sentence with a capital letter and end with a period.

Многострочные комментарии: используйте соответствующие скобки (`--[[ ]]--`) вместо простых (`--[[ ]]`).

Public function comments:

```
--- Copy any table (shallow and deep version).
-- * deepcopy: copies all levels
-- * shallowcopy: copies only first level
-- Supports __copy metamethod for copying custom tables with metatables.
-- @function gsplit
-- @table         inp  original table
-- @shallow[opt]  sep  flag for shallow copy
-- @returns            table (copy)
```

### Тестирование

Use the `tap` module for writing efficient tests. Example of a test file:

```
#!/usr/bin/env tarantool

local test = require('tap').test('table')
test:plan(31)

do
    -- Check basic table.copy (deepcopy).
    local example_table = {
        { 1, 2, 3 },
        { "help, I'm very nested", { { { } } } }
    }

    local copy_table = table.copy(example_table)

    test:is_deeply(
            example_table,
            copy_table,
            "checking, that deepcopy behaves ok"
    )
    test:isnt(
            example_table,
            copy_table,
            "checking, that tables are different"
    )
    test:isnt(
            example_table[1],
            copy_table[1],
            "checking, that tables are different"
    )
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    test:isnt(
            example_table[2],
            copy_table[2],
            "checking, that tables are different"
    )
    test:isnt(
            example_table[2][2],
            copy_table[2][2],
            "checking, that tables are different"
    )
    test:isnt(
            example_table[2][2][1],
            copy_table[2][2][1],
            "checking, that tables are different"
    )
end

<...>

os.exit(test:check() and 0 or 1)
```

When you test your code, the output will be something like this:

```
TAP version 13
1..31
ok - checking, that deepcopy behaves ok
ok - checking, that tables are different
ok - checking, that tables are different
ok - checking, that tables are different
ok - checking, that tables are different
ok - checking, that tables are different
...
```

### Error handling

Принимайте разнообразные значения и выдавайте строго определенные.

With error handling, this means that you must provide an error object as the second multi-return value in case of error. The error object can be a string, a Lua table, cdata, or userdata. In the latter three cases, it must have a `__tostring` metamethod defined.

В случае ошибки нулевое значение `nil` должно быть первым возвращаемым значением. В таком случае ошибку трудно игнорировать.

При проверке возвращаемых значений функции проверяйте сначала первый аргумент. Если это `nil`, ищите ошибку во втором аргументе:

```
local data, err = foo()
if data == nil then
    return nil, err
end
return bar(data)
```

Unless the performance of your code is paramount, try to avoid using more than two return values.

In rare cases, you may want to return `nil` as a legal return value. In this case, it's OK to check for error first and then for return:

```
local data, err = foo()
if err == nil then
    return data
end
return nil, err
```

### luacheck

Для проверки стиля кода Tarantool использует `luacheck`. Он анализирует различные аспекты кода, например неиспользуемые переменные, и иногда проверяет больше аспектов, чем нужно. Поэтому есть соглашение игнорировать некоторые предупреждения, которые выдает `luacheck`:

```
"212/self",   -- Неиспользуемый аргумент <self>.
"411",        -- Переопределение локальной переменной.
"421",        -- Переопределение локальной переменной на вложенном уровне.
"431",        -- Переопределение значения, заданного ранее, в функции.
"432",        -- Перегрузка аргумента на вложенном уровне.
```

# ГЛАВА 11

# Примечания к версиям

This section contains information about Tarantool releases: release notes, lifecycle information, release policy, and other documents. To download Tarantool releases, check the Download page.

All currently supported versions are listed on this page *below*. The information about earlier versions is provided in *EOL versions*.

The Enterprise Edition of Tarantool is distributed in the form of an SDK that has its own versioning. See the *Enterprise SDK changelog* to learn about SDK version numbering and changes.

The detailed information about Tarantool version numbering and release lifecycle is available in *Tarantool release policy*.

Backward compatibility is guaranteed between all versions in the same *release series*. It is also appreciated but not guaranteed between different release series (major number changes). To learn more, read the *Compatibility guarantees* article.

## 11.1 Supported versions

Every Tarantool release series has *the same lifecycle* defined by the release policy. The following diagram visualizes the lifecycle of currently supported Tarantool versions:

The table below provides information about supported versions with links to their *What's new* pages in the documentation and detailed changelogs on GitHub. For information about earlier versions, see *EOL versions*.

---

**Примечание:** *End of life* (*EOL*) means the release series will no longer receive any patches, updates, or feature improvements after the specified date. Versions that haven't reached their end of life yet are shown in **bold**.

*End of support* (*EOS*) means that we won't provide technical support to product versions after the specified date.

---

| Series | First release date | End of life | End of support | Versions |
|---|---|---|---|---|
| *3.1* | **April 16, 2024** | **April 16, 2026** | **Not planned yet** | v. 3.1.0 |
| *3.0* | **December 26, 2023** | **December 26, 2025** | **Not planned yet** | v. 3.0.2<br>v. 3.0.1<br>v. 3.0.0 |
| *2.11 LTS* | **May 24, 2023** | **May 24, 2025** | **Not planned yet** | v. 2.11.3<br>v. 2.11.2<br>v. 2.11.1<br>v. 2.11.0 |
| 2.10 | May 22, 2022 | September 14, 2023 | Not planned yet | *2.10.8*<br>*2.10.7*<br>*2.10.6*<br>*2.10.5*<br>*2.10.4*<br>*2.10.3*<br>*2.10.2*<br>*2.10.1*<br>*2.10.0* |
| 2.8 | August 19, 2021 | April 25, 2022 | December 31, 2024 | *2.8.4*<br>*2.8.3*<br>*2.8.2* |

### 11.1.1 Tarantool 3.1

Release date: April 16, 2024

Releases on GitHub: v. 3.1.0

The 3.1 release of Tarantool continues the development of a new cluster configuration approach introduced in the *3.0 version* and adds the following main product features and improvements for the Community and Enterprise editions:

- **Community Edition (CE)**
  - Improved developer experience for handling errors using the `box.error` module.
  - Introduced fixed-size numeric field types: `uint8`, `int8`, `uint16`, and more.
  - Added RPC functionality for accessing custom roles from the configuration.

  - Made the `tt` utility used to manage instances fully compatible with the latest Tarantool version.

- **Enterprise Edition (EE)**
  - Introduced an external coordinator for automatic and manual failover.
  - Improved the stability of work with the centralized configuration stored in etcd.

### Developing applications

#### Error handling

This release improves the developer experience for handling errors using the *box.error* module. Below are listed the most notable features and changes.

#### Error payload fields

With the 3.1 release, you can add a custom payload to an error. The payload is passed as key-value pairs where a key is a string and a value is any Lua object. In the example below, the `description` key is used to keep the custom payload.

```
custom_error = box.error.new({ type = 'CustomInternalError',
                               message = 'Internal server error',
                               description = 'Some error details'   -- payload
})
```

A payload field value can be accessed using the dot syntax:

```
tarantool> custom_error.description
---
- Some error details
...
```

#### Error stacks

The 3.1 release simplifies creating error chains. In the earlier versions, you need to set an error cause using the *set_prev(error_object)* method, for example:

```
local ok, err = pcall(my_func)
if not ok then
    local err2 = box.error.new{type = "MyAppError", message = "my_func failed"}
    err2:set_prev(err)
    err2:raise()
end
```

Using this approach, you need to construct a new error without raising it, then set its cause using `set_prev()`, and only then raise it. Starting with the 3.1 version, you can use a new `prev` argument when constructing an error:

```
local ok, err = pcall(my_func)
if not ok then
    box.error{type = "MyAppError", message = "my_func failed", prev = err}
end
```

### Error serialization improvements

The 3.1 release allows you to increase the verbosity of error serialization. Before the 3.1 release, a serialized error representation included only an error message:

```
tarantool> box.error.new({ type = 'CustomInternalError', message = 'Internal server error
↪'})
---
- Internal server error
...
```

Starting with the 3.1 version, a serialized error also includes other fields that might be useful for analyzing errors:

```
tarantool> box.error.new({ type = 'CustomInternalError', message = 'Internal server error
↪'})
---
- code: 0
  base_type: CustomError
  type: CustomInternalError
  custom_type: CustomInternalError
  message: Internal server error
  trace:
  - file: '[C]'
    line: 4294967295
...
```

Logging an error using a built-in *logging module* prints an error message followed by a tab space (\t) and all the payload fields serialized as a JSON map, for example:

```
main/104/app.lua/tarantool I> Internal server error {"code":0,"base_type":"CustomError",
↪"type":"CustomInternalError", ... }
```

Given that this change may change the behavior of existing code, a new `box_error_serialize_verbose` *compat* option is introduced. To try out an increased verbosity of error serialization, set this option to `new`:

```
tarantool> require('compat').box_error_serialize_verbose = 'new'
---
...
```

### Fixed-size numeric types

The 3.1 release introduces fixed-size numeric *types* that might be useful to store data unencoded in an array for effective scanning. The following numeric types are added:

- `uint8`: an integer in a range `[0 .. 255]`.
- `int8`: an integer in a range `[-128 .. 127]`.
- `uint16`: an integer in a range `[0 .. 65,535]`.
- `int16`: an integer in a range `[-32,768 .. 32,767]`.
- `uint32`: an integer in a range `[0 .. 4,294,967,295]`.
- `int32`: an integer in a range `[-2,147,483,648 .. 2,147,483,647]`.

- `uint64`: an integer in a range `[0 .. 18,446,744,073,709,551,615]`.
- `int64`: an integer in a range `[-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807]`.
- `float32`: a 32-bit floating point number.
- `float64`: a 64-bit floating point number.

### Experimental „connpool“ module

A new `experimental.connpool` *module* provides a set of features for remote connections to any cluster instance or executing remote procedure calls on an instance that meets the specified criteria. To load the `experimental.connpool` module, use the `require()` directive:

```
sharded_cluster:router-a-001> connpool = require('experimental.connpool')
---
...
```

In the 3.1 version, this module provides the following API:

- The `connect()` function accepts an instance name and returns the active connection to this instance:

  ```
  sharded_cluster:router-a-001> conn = connpool.connect("storage-b-002")
  ---
  ...
  ```

  Once you have a connection, you can execute requests on a remote instance, for example, select data from a space:

  ```
  sharded_cluster:router-a-001> conn.space.bands:select({}, { limit = 5 })
  ---
  - - [3, 804, 'Ace of Base', 1987]
    - [7, 693, 'The Doors', 1965]
    - [9, 644, 'Led Zeppelin', 1968]
    - [10, 569, 'Queen', 1970]
  ...
  ```

- The `filter()` function returns the names of instances that match the specified conditions. In the example below, this function returns a list of instances with the `storage` role and specified label value:

  ```
  sharded_cluster:router-a-001> connpool.filter({ roles = { 'storage' }, labels = {␣
  ↪dc = 'east' }})
  ---
  - - storage-b-002
    - storage-a-002
  ...
  ```

- The `call()` function can be used to execute a function on a remote instance. In the example below, the following conditions are specified to choose an instance to execute the `vshard.storage.buckets_count` function on:
  - An instance has the `storage` role.
  - An instance has the `dc` label set to `west`.
  - An instance is writable.

```
sharded_cluster:router-a-001> connpool.call('vshard.storage.buckets_count', nil, {␣
↪roles = { 'storage' }, labels = { dc = 'west' }, mode = 'rw' })
---
- 500
...
```

### Accessing configuration of other cluster members

In Tarantool 3.0, the *config* module provides the ability to work with a current instance's configuration only. Starting with the 3.1 version, you can get all the instances that constitute a cluster and obtain the configuration of any instance of this cluster.

The `config:instances()` function lists all instances of the cluster:

```
sharded_cluster:router-a-001> require('config'):instances()
---
- storage-a-001:
    group_name: storages
    instance_name: storage-a-001
    replicaset_name: storage-a
  storage-b-002:
    group_name: storages
    instance_name: storage-b-002
    replicaset_name: storage-b
  router-a-001:
    group_name: routers
    instance_name: router-a-001
    replicaset_name: router-a
  storage-a-002:
    group_name: storages
    instance_name: storage-a-002
    replicaset_name: storage-a
  storage-b-001:
    group_name: storages
    instance_name: storage-b-001
    replicaset_name: storage-b
...
```

To get the specified configuration value for a certain instance, pass an instance name as an argument to `config:get()`:

```
sharded_cluster:router-a-001> require('config'):get('iproto', {instance = 'storage-b-001
↪'})
---
- readahead: 16320
  net_msg_max: 768
  listen:
  - uri: 127.0.0.1:3304
  threads: 1
  advertise:
    peer:
      login: replicator
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    client: null
    sharding:
      login: storage
...
```

### Administration and maintenance

#### Failover coordinator (EE)

Tarantool Enterprise Edition 3.1 introduces an external failover coordinator that monitors a Tarantool cluster and performs automatic leadership change if a current replica set leader is inaccessible.

A failover coordinator requires the *replication.failover* configuration option to be set to `supervised`:

```
replication:
  failover: supervised

# ...
```

To start a failover coordinator, execute the `tarantool` command with the `failover` option and pass a path to a *YAML configuration file*:

```
$ tarantool --failover --config /path/to/config
```

A failover coordinator connects to all the instances, polls them for their status, and controls that each replica set with `replication.failover` set to `supervised` has only one writable instance.

Optionally, you can configure failover timeouts and other parameters in the `failover` section at the *global level*:

```
failover:
  call_timeout: 1
  lease_interval: 15
  renew_interval: 5
  stateboard:
    renew_interval: 1
    keepalive_interval: 5
```

#### Sharding

The 3.1 release includes new *sharding* options that provide additional flexibility for configuring a sharded cluster. A new `sharding.weight` specifies the relative amount of data that a replica set can store. In the example below, the `storage-a` replica set can store twice as much data as `storage-b`:

```
# ...
replicasets:
  storage-a:
    sharding:
      weight: 2
    # ...
  storage-b:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  sharding:
    weight: 1
  # ...
```

The `sharding.rebalancer_mode` option configures whether a rebalancer is selected manually or automatically. This option can have one of three values:

- `auto` (default): if there are no replica sets with the `rebalancer` sharding role (*sharding.roles*), a replica set with the rebalancer will be selected automatically among all replica sets.
- `manual`: one of the replica sets should have the `rebalancer` sharding role. The rebalancer will be in this replica set.
- `off`: rebalancing is turned off regardless of whether a replica set with the `rebalancer` sharding role exists or not.

#### Compatibility with the tt utility

With this release, the `tarantoolctl` utility used to administer Tarantool instances is completely removed from Tarantool packages. The latest version of the *tt utility* is fully compatible with Tarantool 3.1 and covers all the required functionality:

- Setting up a development environment: initializing the environment and installing different Tarantool versions.
- Various capabilities for developing cluster applications: creating applications from templates, managing modules, and building and packaging applications.
- Managing cluster instances: starting and stopping instances, connecting to remote instances for administration, and so on.
- Importing and exporting data (Enterprise Edition only).

Learn how to migrate from `tarantoolctl` to `tt` in the *Migration from tarantoolctl to tt* section.

### 11.1.2 Tarantool 3.0

Release date: December 26, 2023

Releases on GitHub: v. 3.0.2, v. 3.0.1, v. 3.0.0

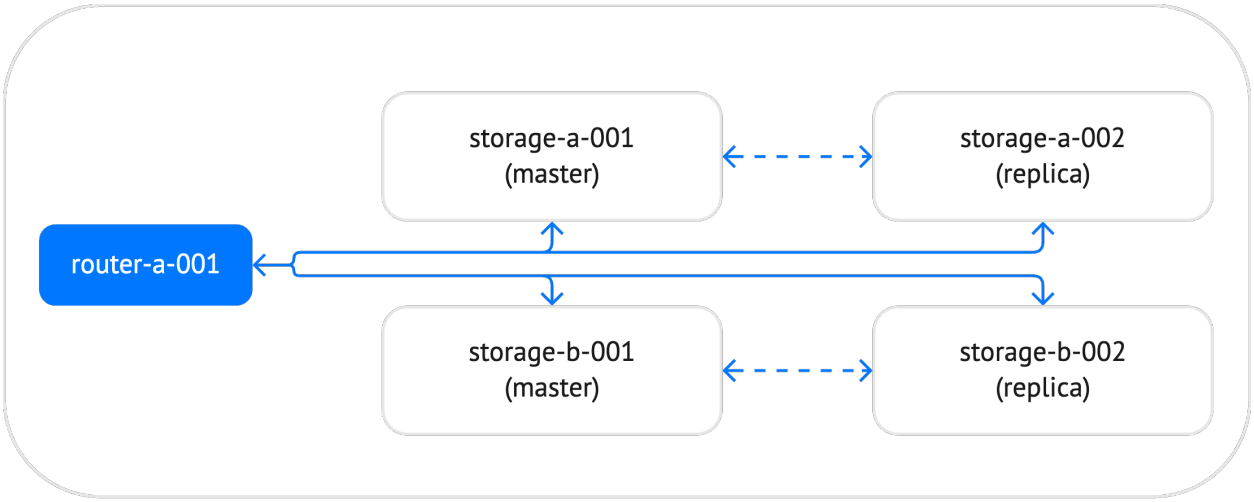
The 3.0 release of Tarantool introduces a new declarative approach for configuring a cluster, a new visual tool – Tarantool Cluster Manager, and many other new features and fixes. This document provides an overview of the most important features for the Community and Enterprise editions.

- *New declarative configuration*
- *Tarantool Cluster Manager*
- *Administration and maintenance*
- *Developing applications*
- *Stability*

### New declarative configuration

Starting with the 3.0 version, Tarantool provides the ability to *configure the full topology of a cluster* using a declarative YAML configuration instead of configuring each instance using a dedicated Lua script. With a new approach, you can write a local configuration in a YAML file for each instance or store configuration data in one reliable place, for example, a Tarantool or an etcd cluster.

The example below shows how a configuration of a small sharded cluster might look. In the diagram below, the cluster includes 5 instances: one router and 4 storages, which constitute two replica sets. For each replica set, the master instance is specified manually.




The example below demonstrates how a topology of such a cluster might look in a YAML configuration file:

```
groups:
  storages:
    app:
      module: storage
    sharding:
      roles: [storage]
    replication:
      failover: manual
    replicasets:
      storage-a:
        leader: storage-a-001
        instances:
          storage-a-001:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
          storage-a-002:
            iproto:
              listen:
              - uri: '127.0.0.1:3303'
      storage-b:
        leader: storage-b-001
        instances:
          storage-b-001:
            iproto:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            listen:
            - uri: '127.0.0.1:3304'
        storage-b-002:
          iproto:
            listen:
            - uri: '127.0.0.1:3305'
  routers:
    app:
      module: router
    sharding:
      roles: [router]
    replicasets:
      router-a:
        instances:
          router-a-001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
```

You can find the full sample in the GitHub documentation repository: sharded_cluster.

The latest version of the tt utility provides the ability to *manage Tarantool instances* configured using a new approach. You can start all instances in a cluster by executing one command, check the status of instances, or stop them:

```
$ tt start sharded_cluster
   • Starting an instance [sharded_cluster:storage-a-001]...
   • Starting an instance [sharded_cluster:storage-a-002]...
   • Starting an instance [sharded_cluster:storage-b-001]...
   • Starting an instance [sharded_cluster:storage-b-002]...
   • Starting an instance [sharded_cluster:router-a-001]...
```

### Centralized configuration (EE)

Tarantool Enterprise Edition enables you to store configuration data in one reliable place, for example, an *etcd* cluster. To achieve this, you need to configure connection options in the `config.etcd` section of the configuration file, for example:

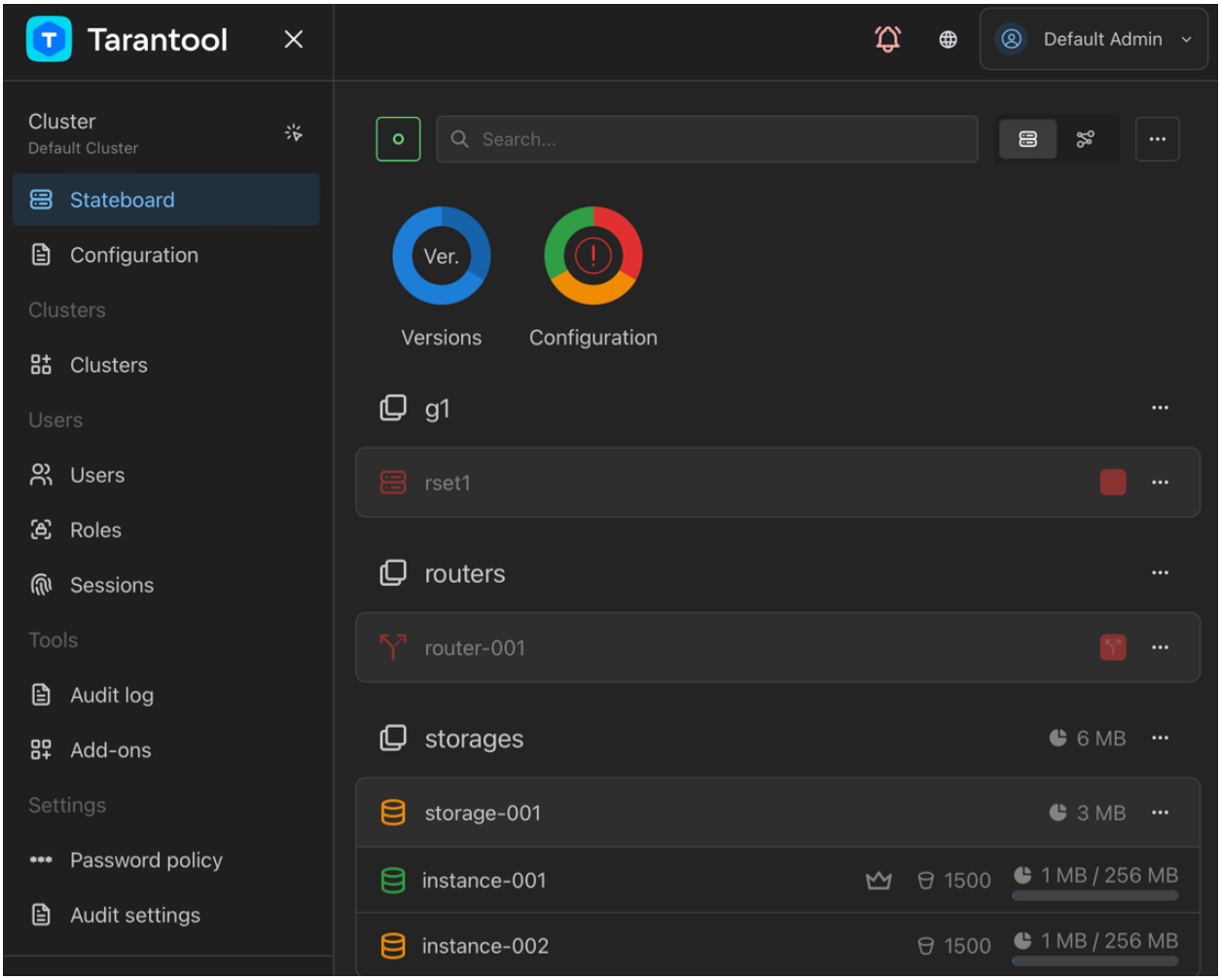
```
config:
  etcd:
    endpoints:
    - http://localhost:2379
    prefix: /myapp
    username: sampleuser
    password: '123456'
```

Using the configuration above, a Tarantool instance searches for a cluster configuration by the following path:

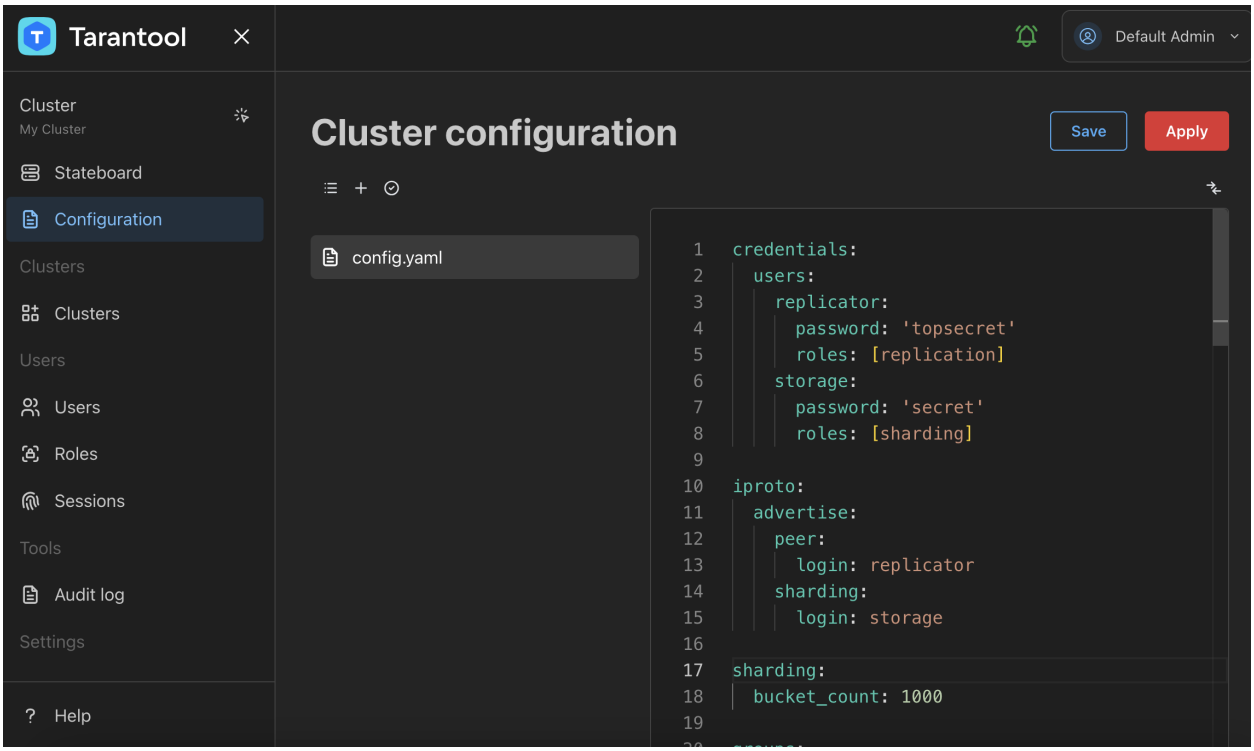
```
http://localhost:2379/myapp/config/*
```

### Tarantool Cluster Manager (EE)

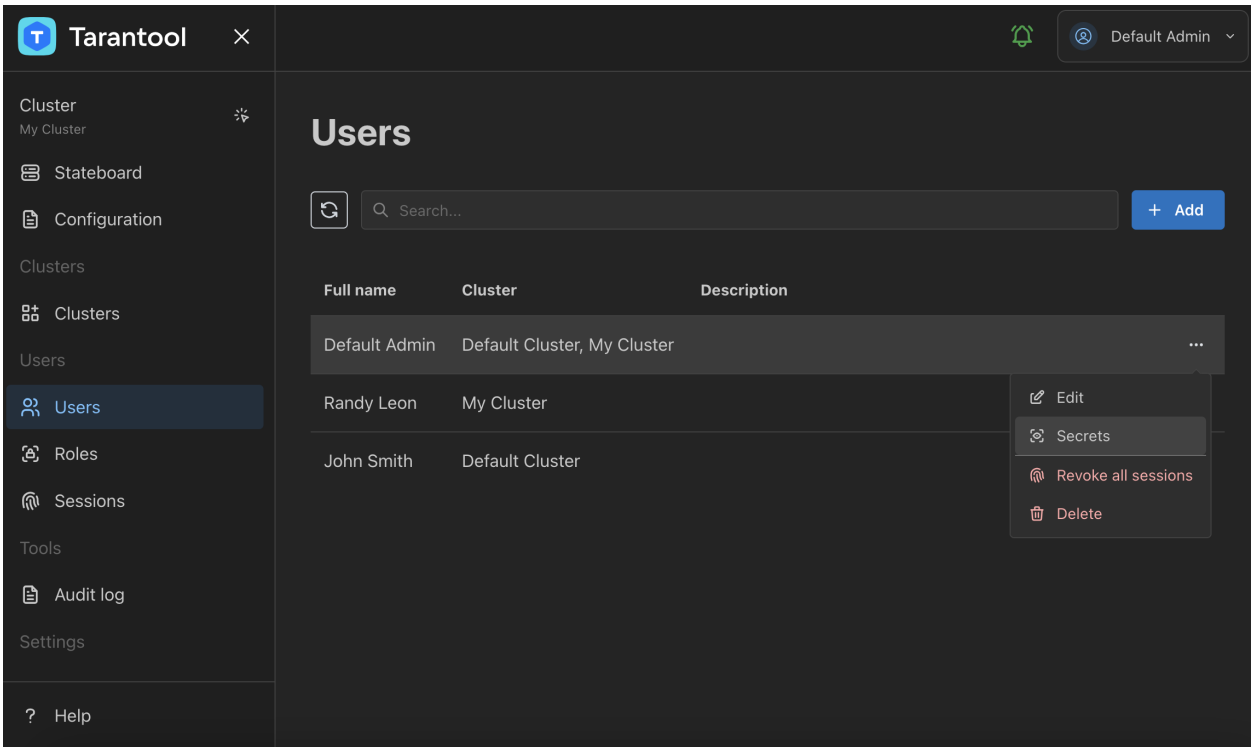
Tarantool 3.0 Enterprise Edition comes with a brand new visual tool – *Tarantool Cluster Manager* (TCM). It provides a web-based user interface for managing, configuring, and monitoring Tarantool EE clusters that use centralized configuration storage.



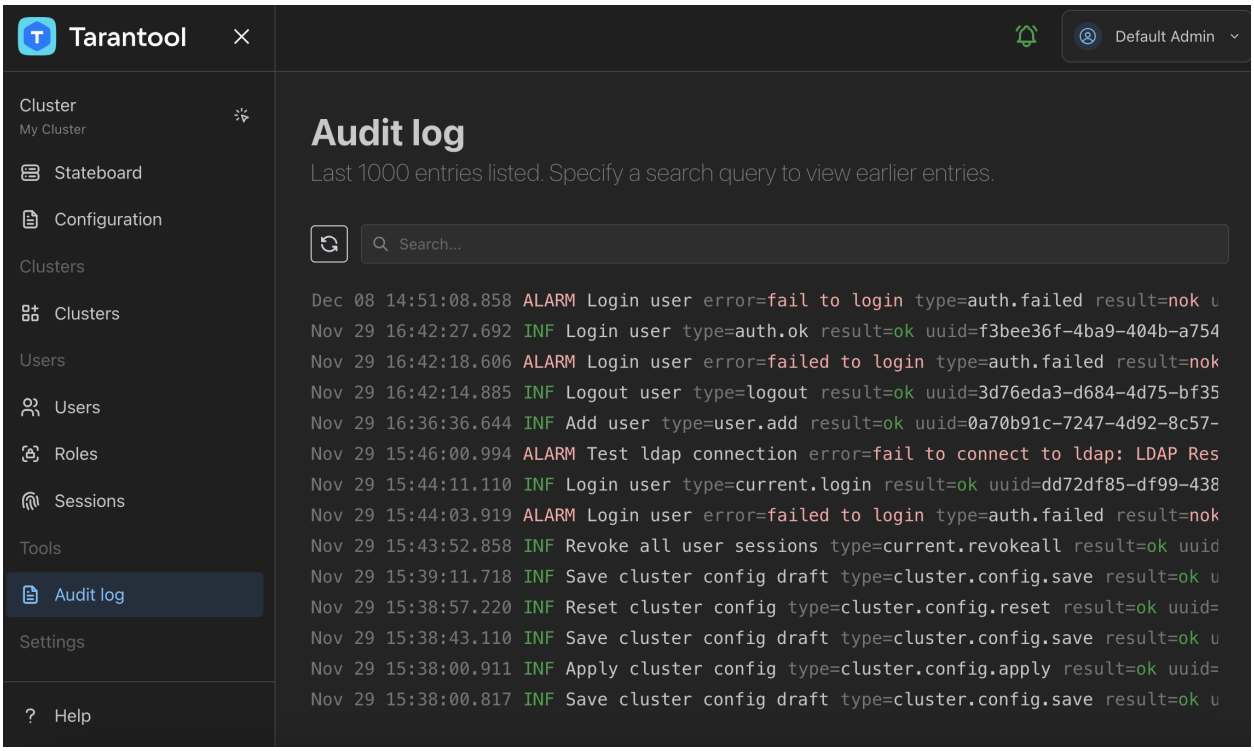

TCM can manage multiple clusters and covers a wide range of tasks, from writing a cluster's configuration to executing commands interactively on specific instances.




TCM's role-based access control system lets you manage users' access to clusters, their configurations, and stored data.




The built-in customizable audit logging mechanism and LDAP authentication make TCM a suitable solution for different enterprise security requirements.




### Administration and maintenance

### Database statistics

Starting with 3.0, Tarantool provides extended statistics about memory consumption for the given space or specific tuples.

Usually, the *space_object:bsize()* method is used to get the size of memory occupied by the specified space:

```
app:instance001> box.space.books:bsize()
---
- 70348673
...
```

In addition to the actual data, the space requires additional memory to store supplementary information. You can see the total memory usage using *box.slab.info()*:

```
app:instance001> box.slab.info().items_used
---
- 75302024
...
```

A new `space_object:stat()` method allows you to determine how the additional 5 Mb of memory is used:

```
app:instance001> box.space.books:stat()
---
- tuple:
    memtx:
      waste_size: 1744011
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      data_size: 70348673
      header_size: 2154132
      field_map_size: 0
    malloc:
      waste_size: 0
      data_size: 0
      header_size: 0
      field_map_size: 0
...
```

The above report gives the following information:

- `header_size` and `field_map_size`: the size of service information.
- `data_size`: the actual size of data, which equals to `space_object:bsize()`.
- `waste_size`: the size of memory wasted due to internal fragmentation in the slab allocator.

To get such information about a specific tuple, use `tuple_object:info()`:

```
app:instance001> box.space.books:get('1853260622'):info()
---
- data_size: 277
  waste_size: 9
  arena: memtx
  field_map_size: 0
  header_size: 10
...
```

### Bootstrapping a replica set

The new version includes the capability to choose a bootstrap leader for a replica set manually. The bootstrap leader is a node that creates an initial snapshot and registers all the replicas in a replica set.

First, you need to set *replication.bootstrap_strategy* to `config`. Then, use the `<replicaset_name>.bootstrap_leader` option to specify a bootstrap leader.

```
groups:
  group001:
    replicasets:
      replicaset001:
        replication:
          bootstrap_strategy: config
        bootstrap_leader: instance001
        instances:
          instance001:
            iproto:
              listen:
              - uri: '127.0.0.1:3301'
          instance002:
            iproto:
              listen:
              - uri: '127.0.0.1:3302'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        instance003:
          iproto:
            listen:
            - uri: '127.0.0.1:3303'
```

**Примечание:** Note that in 3.0, the *replication_connect_quorum* option is removed. This option was used to specify the number of nodes to be up and running for starting a replica set.

### Security (EE)

With the 3.0 version, Tarantool Enterprise Edition provides a set of new features that enhance *security* in your cluster:

- Introduced the `secure_erasing` configuration option that forces Tarantool to overwrite a data file a few times before deletion to render recovery of a deleted file impossible. With the new configuration approach, you can enable this capability as follows:

  ```
  security:
    secure_erasing: true
  ```

  This option can be also set using the `TT_SECURITY_SECURE_ERASING` environment variable.

- Added the `auth_retries` option that configures the maximum number of authentication retries before throttling is enabled. You can configure this option as follows:

  ```
  security:
    auth_retries: 3
  ```

- Added the capability to use the new SSL certificate with the same name by reloading the configuration. To do this, use the `reload()` function provided by the new `config` module:

  ```
  app:instance001> require('config'):reload()
  ---
  ...
  ```

### Audit logging (EE)

Tarantool Enterprise Edition includes the following new features for *audit logging*:

- Added a unique identifier (UUID) to each audit log entry.
- Introduced audit log severity levels. Each system audit event now has a severity level determined by its importance.
- Added the `audit_log.audit_spaces` option that configures the list of spaces for which data operation events should be logged.
- Added the `audit_log.audit_extract_key` option that forces the audit subsystem to log the primary key instead of a full tuple in DML operations. This might be useful for reducing audit log size in the case of large tuples.

The sample audit log configuration in the 3.0 version might look as follows, including new `audit_spaces` and `audit_extract_key` options:

```
audit_log:
  to: file
  file: audit_tarantool.log
  filter: [ddl,dml]
  spaces: [books]
  extract_key: true
```

With this configuration, an audit log entry for a DELETE operation may look like below:

```
{
  "time": "2023-12-19T10:09:44.664+0000",
  "uuid": "65901190-f8a6-45c1-b3a4-1a11cf5c7355",
  "severity": "VERBOSE",
  "remote": "unix/:(socket)",
  "session_type": "console",
  "module": "tarantool",
  "user": "admin",
  "type": "space_delete",
  "tag": "",
  "description": "Delete key [\"0671623249\"] from space books"
}
```

The entry includes the new `uuid` and `severity` fields. The last `description` field gives only the information about the key of the deleted tuple.

### Reading flight recordings (EE)

The *flight recorder* available in the Enterprise Edition is an event collection tool that gathers various information about a working Tarantool instance. With the 3.0 version, you can read flight recordings using the API provided by the `flightrec` module.

To enable the flight recorder in a YAML file, set `flightrec.enabled` to true:

```
flightrec:
  enabled: true
```

Then, you can use the Lua API to open and read `*.ttfr` files:

```
app:instance001> flightrec = require('flightrec')
---
...

app:instance001> flightrec_file = flightrec.open('var/lib/instance001/20231225T085435.
↪ttfr')
---
...

app:instance001> flightrec_file
---
- sections: &0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    requests:
      size: 10485760
    metrics:
      size: 368640
    logs:
      size: 10485760
  was_closed: false
  version: 0
  pid: 1350
...

app:instance001> for i, r in flightrec_file.sections.logs:pairs() do record = r; break␣
↪end
---
...

app:instance001> record
---
- level: INFO
  fiber_name: interactive
  fiber_id: 103
  cord_name: main
  file: ./src/box/flightrec.c
  time: 2023-12-25 08:50:12.275
  message: 'Flight recorder: configuration has been done'
  line: 727
...

app:instance001> flightrec_file:close()
---
...
```

### New DEB and RPM packages

With this release, the approach to delivering Tarantool to end users in DEB and RPM packages is slightly revised. In the previous versions, Tarantool was built for the most popular Linux distributions and their latest version.

Starting with this release, only two sets of DEB and RPM packages are delivered. The difference is that these packages include a statically compiled Tarantool binary. This approach provides the ability to install DEB and RPM packages on any Linux distributions that are based on CentOS and Debian.

To ensure that Tarantool works for a wide range of different distributions and their versions, RPM and DEB packages are prepared on CentOS 7 with glibc 2.17.

### Developing applications

#### varbinary in Lua

In the previous versions, Tarantool already supported the `varbinary` type for *storing data*. But working with `varbinary` database fields required workarounds, such as using C to process such data.

The 3.0 version includes a new `varbinary` module for working with varbinary objects. The module implements the following functions:

- `varbinary.new()` - constructs a varbinary object from a plain string.
- `varbinary.is()` - returns true if the argument is a varbinary object.

In the example below, an object is created from a string:

```
local varbinary = require('varbinary')
local bin = varbinary.new('Hello world!')
```

The built-in decoders now decode binary data fields to a varbinary object by default:

```
local varbinary = require('varbinary')
local msgpack = require('msgpack')
varbinary.is(msgpack.decode('\xC4\x02\xFF\xFE'))
--[[
---
- true
...
]]
varbinary.is(yaml.decode('!!binary //4='))
--[[
---
- true
...
]]
```

This also implies that the data stored in the database under the `varbinary` field type is now returned to Lua not as a plain string but as a varbinary object.

It's possible to revert to the old behavior by toggling the new `binary_data_decoding` *compat* option because this change may break backward compatibility:

```
compat:
  binary_data_decoding: old
```

#### Default field values

You can now assign the default values for specific fields when defining a *space format*. In this example, the `isbn` and `title` fields have the specified default values:

```
box.schema.space.create('books')
box.space.books:format({
    { name = 'id', type = 'unsigned' },
    { name = 'isbn', type = 'string', default = '9990000000000' },
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    { name = 'title', type = 'string', default = 'New awesome book' },
    { name = 'year_of_publication', type = 'unsigned', default = 2023 }
})
box.space.books:create_index('primary', { parts = { 'isbn' } })
```

If you insert a tuple with missing fields, the default values are inserted:

```
app:instance001> box.space.books:insert({ 1000, nil, nil, nil })
---
- [1000, '9990000000000', 'New awesome book', 2023]
...
```

You can also provide a custom logic for generating a default value. To achieve this, create a function using `box.schema.func.create`:

```
box.schema.func.create('current_year', {
    language = 'Lua',
    body = "function() return require('datetime').now().year end"
})
```

Then, assign the function name to `default_func` when defining a space format:

```
box.space.books:format({
    -- ... --
    { name = 'year_of_publication', type = 'unsigned', default_func = 'current_year' }
})
```

### Triggers

In the 3.0 version, the API for creating *triggers* is completely reworked. A new `trigger` module is introduced, allowing you to set handlers on both predefined and custom events.

To create the trigger, you need to:

1) Provide an event name used to associate the trigger with.
2) Define the trigger name.
3) Provide a trigger handler function.

The code snippet below shows how to subscribe to changes in the `books` space:

```
local trigger = require('trigger')
trigger.set(
        'box.space.books.on_replace', -- event name
        'some-custom-trigger',        -- trigger name
        function(...)
            -- trigger handler
        end
)
```

### Pagination in read views (EE)

The 2.11 release introduced the following features:

- *Read views* are in-memory snapshots of the entire database that aren't affected by future data modifications.
- *Pagination* for getting data in chunks.

With the 3.0 release, a read view object supports the `after` and `fetch_pos` arguments for the `select` and `pairs` methods:

```
-- Select first 3 tuples and fetch a last tuple's position --
app:instance001> result, position = read_view1.space.bands:select({}, { limit = 3, fetch_
↪pos = true })
---
...

app:instance001> result
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'Ace of Base', 1987]
...

app:instance001> position
---
- kQM
...

-- Then, you can pass this position as the 'after' parameter --
app:instance001> read_view1.space.bands:select({}, { limit = 3, after = position })
---
- - [4, 'The Beatles', 1960]
  - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
...
```

### IPROTO tuple format

Starting with the 3.0 version, the IPROTO protocol is extended to support for sending names of tuple fields in the *IPROTO_CALL* and other IPROTO responses. This simplifies the development of Tarantool connectors and also simplifies handling tuples received from remote procedure calls or from routers.

It's possible to revert to the old behavior by toggling the `box_tuple_extension` *compat* option:

```
compat:
  box_tuple_extension: old
```

#### SQL: case-sensitive names

Starting with 3.0, names in *SQL*, for example, table, column, or constraint names are case-sensitive. Before the 3.0 version, the query below created a `MYTABLE` table:

```
CREATE TABLE MyTable (i INT  PRIMARY KEY);
```

To create the `MyTable` table, you needed to enclose the name into double quotes:

```
CREATE TABLE "MyTable" (i INT  PRIMARY KEY);
```

Starting with 3.0, names are case-sensitive, and double quotes are no longer needed:

```
CREATE TABLE MyTable (i INT  PRIMARY KEY);
```

For backward compatibility, the new version also supports a second lookup using an uppercase name. This means that the query below tries to find the `MyTable` table and then `MYTABLE`:

```
SELECT * FROM MyTable;
```

### Stability

#### Handling LuaJIT compiler errors

The 3.0 release includes a fix for the gh-562 LuaJIT issue related to the inability to handle internal compiler on-trace errors using `pcall`. The examples of such errors are:

- An `Out of memory` error might occur for `select` queries returning a large amount of data.
- A `Table overflow` error is raised when exceeding the maximum number of keys in a table.

The script below tries to fill a Lua table with a large number of keys:

```
local function memory_payload()
    local t = {}
    for i = 1, 1e10 do
        t[ffi.new('uint64_t')] = i
    end
end
local res, err = pcall(memory_payload)
print(res, err)
```

In the previous Tarantool version with the 32-bit Lua GC, this script causes the following error despite using `pcall`:

```
PANIC: unprotected error in call to Lua API (not enough memory)
```

For Tarantool with the 64-bit Lua GC, this script causes a `Table overflow` error:

```
PANIC: unprotected error in call to Lua API (table overflow)
```

Starting with the 3.0 version, these errors are handled correctly with the following outputs:

```
false    not enough memory -- 32-bit Lua GC
false    table overflow    -- 64-bit Lua GC
```

As a result, Tarantool 3.0 becomes more stable in cases when user scripts include erroneous code.

### 11.1.3 Tarantool 2.11 (LTS)

Release date: May 24, 2023

Releases on GitHub: v. 2.11.3, v. 2.11.2, v. 2.11.1, v. 2.11.0

The 2.11 release of Tarantool includes many new features and fixes. This document provides an overview of the most important features for the Enterprise and Community editions.

2.11 is the long-term support (LTS) release with two years of maintenance. This means that you will receive all the necessary security fixes and bug fixes throughout this period, and be able to get technical support afterward. You can learn more about the Tarantool release policy from the *corresponding document*.

| *Enterprise Edition* | *Community Edition* |
|---|---|
| • *Security enhancements*<br>• *Read views*<br>• *Data compression improvements*<br>• *WAL extensions* | • *Pagination*<br>• *Downgrading a database*<br>• *New bootstrap strategy*<br>• *Limitation of fiber execution time*<br>• *Per-module logging*<br>• *HTTP client enhancements*<br>• *Linearizable read*<br>• *Explicit sequential scanning in SQL*<br>• *Strict fencing in RAFT* |

#### Upgrading to Tarantool 2.11

Tarantool provides the live upgrade mechanism that enables cluster upgrade without downtime. In case of upgrade issues, you can roll back to the original state without downtime as well.

To learn how to upgrade to Tarantool 2.11, see *Upgrades*.

#### Enterprise Edition

#### Security enhancements

#### Encrypted SSL/TLS keys

Tarantool Enterprise Edition now supports encrypted SSL/TLS private key files protected with a password. Given that most certificate authorities generate encrypted keys, this feature simplifies the maintenance of Tarantool instances.

A password can be provided using either the new `ssl_password` URI parameter or in a text file specified using `ssl_password_file`, for example:

```
box.cfg{ listen = {
    uri = 'localhost:3301',
    params = {
        transport = 'ssl',
        ssl_key_file = '/path_to_key_file',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        ssl_cert_file = '/path_to_cert_file',
        ssl_ciphers = 'HIGH:!aNULL',
        ssl_password = 'topsecret'
    }
}}
```

To learn more, see *Traffic encryption*.

### Security enforcement options

With 2.11, Tarantool Enterprise Edition includes new security enforcement options. These options enable you to enforce the use of strong passwords, set up a maximum password age, and so on. For example, the `password_min_length` configuration option specifies the minimum number of characters for a password:

```
box.cfg{ password_min_length = 10 }
```

To specify the maximum period of time (in days) a user can use the same password, you can use the `password_lifetime_days` option, which uses the system clock under the hood:

```
box.cfg{ password_lifetime_days = 365 }
```

Note that by default, new options are not specified. You can learn more about all the available options from the *Authentication restrictions* and *Password policy* sections.

### PAP-SHA256 authentication method

By default, Tarantool uses the CHAP protocol to authenticate users and applies SHA-1 hashing to passwords. In this case, password hashes are stored in the `_user` space unsalted. If an attacker gains access to the database, they may crack a password, for example, using a rainbow table.

With the Enterprise Edition, you can enable PAP authentication with the SHA256 hashing algorithm. For PAP, a password is salted with a user-unique salt before saving it in the database.

Given that PAP transmits a password as plain text, Tarantool requires configuring *SSL/TLS*. Then, you need to specify the `box.cfg.auth_type` option as follows:

```
box.cfg{ auth_type = 'pap-sha256' }
```

Learn more from the *Authentication protocol* section.

### Read views

Starting with 2.11, Tarantool Enterprise Edition provides the ability to create read views - in-memory snapshots of the entire database that aren't affected by future data modifications. Read views can be used to make complex analytical queries. This reduces the load on the main database and improves RPS for a single Tarantool instance.

Working with read views consists of three main steps:

1. To create a read view, call the `box.read_view.open()` function:

```
tarantool> read_view1 = box.read_view.open({name = 'read_view1'})
```

2. After creating a read view, you can access database spaces and their indexes and get data using the familiar `select` and `pairs` data-retrieval operations, for example:

```
tarantool> read_view1.space.bands:select({}, {limit = 4})
---
- - [1, 'Roxette', 1986]
  - [2, 'Scorpions', 1965]
  - [3, 'Ace of Base', 1987]
  - [4, 'The Beatles', 1960]
```

3. When a read view is no longer needed, close it using the `read_view_object:close()` method:

```
tarantool> read_view1:close()
```

To learn more, see the *Read views* topic.

### Data compression improvements

Tarantool Enterprise Edition now includes the `zlib` algorithm for *tuple compression*. This algorithm shows good performance in data decompression, which reduces CPU usage if the volume of read operations significantly exceeds the volume of write operations.

To use the new algorithm, set the `compression` option to `zlib` when formatting a space:

```
box.space.my_space:format{
    {name = 'id', type = 'unsigned'},
    {name = 'data', type = 'string', compression = 'zlib'},
}
```

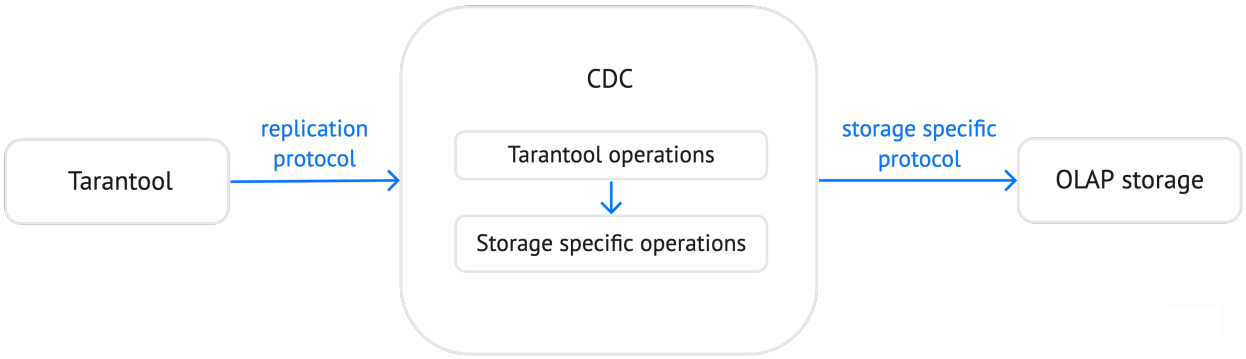
The new *compress* module provides an API for compressing and decompressing arbitrary data strings using the same algorithms available for tuple compression:

```
compressor = require('compress.zlib').new()

data = compressor:compress('Hello world!') -- returns a binary string
compressor:decompress(data) -- returns 'Hello world!'
```

### WAL extensions

Tarantool can use a write-ahead log not only to maintain data persistence and replication. Another use case is implementing a CDC (Change Data Capture) utility that transforms a data replication stream and provides the ability to replicate data from Tarantool to an external storage.




With 2.11, Tarantool Enterprise Edition provides WAL extensions that add auxiliary information to each write-ahead log record. For example, you can enable storing old and new tuples for each write-ahead log record. This is especially useful for the `update` operation because a write-ahead log record contains only a key value.

See the *WAL extensions* topic to learn how to enable and configure WAL extensions.

### Community Edition

### Pagination

With the 2.11 version, Tarantool supports pagination and enables the ability to get data in chunks. The *index_object:select()* and *index_object:pairs()* methods now provide the `after` option that specifies a tuple or a tuple's position after which `select` starts the search. In the example below, the `select` operation gets maximum 3 tuples after the specified tuple:

```
tarantool> bands.index.primary:select({}, {after = {4, 'The Beatles', 1960}, limit = 3})
---
- - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
  - [7, 'The Doors', 1965]
...
```

The `after` option also accepts the position of the tuple represented by the base64 string. For example, you can set the `fetch_pos` boolean option to `true` to return the position of the last selected tuple as the second value:

```
tarantool> result, position = bands.index.primary:select({}, {limit = 3, fetch_pos =␣
↪true})
---
...
```

Then, pass this position as the `after` parameter:

```
tarantool> bands.index.primary:select({}, {limit = 3, after = position})
---
- - [4, 'The Beatles', 1960]
  - [5, 'Pink Floyd', 1965]
  - [6, 'The Rolling Stones', 1962]
...
```

The new `after` and `fetch_pos` options are also implemented by the built-in `net.box` connector. For example, you can use these options to get data *asynchronously*.

### Downgrading a database

The 2.11 version provides the ability to downgrade a database to the specified Tarantool version using the *box.schema.downgrade()* method. This might be useful in the case of a failed upgrade.

To prepare a database for using it on an older Tarantool instance, call `box.schema.downgrade` and pass the desired Tarantool version:

```
tarantool> box.schema.downgrade('2.8.4')
```

To see Tarantool versions available for downgrade, call `box.schema.downgrade_versions()`. The earliest release available for downgrade is *2.8.2*.

### New bootstrap strategy

In previous Tarantool versions, the *replication_connect_quorum* option was used to specify the number of running nodes to start a replica set. This option was designed to simplify a replica set bootstrap. But in fact, this behavior brought some issues during a cluster lifetime and maintenance operations, for example:

- Users who didn't change this option encountered problems with the partial cluster bootstrap.
- Users who changed the option encountered problems during the instance restart.

With 2.11, `replication_connect_quorum` is deprecated in favor of *bootstrap_strategy*. This option works during a replica set bootstrap and implies sensible default values for other parameters based on the replica set configuration. Currently, `bootstrap_strategy` accepts two values:

- `auto`: a node doesn't boot if half or more of the other nodes in a replica set are not connected. For example, if the replication parameter contains 2 or 3 nodes, a node requires 2 connected instances. In the case of 4 or 5 nodes, at least 3 connected instances are required. Moreover, a bootstrap leader fails to boot unless every connected node has chosen it as a bootstrap leader.
- `legacy`: a node requires the `replication_connect_quorum` number of other nodes to be connected. This option is added to keep the compatibility with the current versions of Cartridge and might be removed in the future.

### Limitation of fiber execution time

Starting with 2.11, if a fiber works too long without yielding control, you can use a fiber slice to limit its execution time. The *fiber_slice_default* `compat` option controls the default value of the maximum fiber slice. In future versions, this option will be set to `true` by default.

There are two slice types - a warning and an error slice:

- When a warning slice is over, a warning message is logged, for example:

  ```
  fiber has not yielded for more than 0.500 seconds
  ```

- When an error slice is over, the fiber is cancelled and the `FiberSliceIsExceeded` error is thrown:

  ```
  FiberSliceIsExceeded: fiber slice is exceeded
  ```

Note that these messages can point at issues in the existing application code. These issues can cause potential problems in production.

The fiber slice is checked by all functions operating on spaces and indexes, such as `index_object.select()`, `space_object.replace()`, and so on. You can also use the `fiber.check_slice()` function in application code to check whether the slice for the current fiber is over.

The example below shows how to use `fiber.set_max_slice()` to limit the slice for all fibers. `fiber.check_slice()` is called inside a long-running operation to determine whether a slice for the current fiber is over.

```
-- app.lua --
fiber = require('fiber')
clock = require('clock')

fiber.set_max_slice({warn = 1.5, err = 3})
time = clock.monotonic()
function long_operation()
    while clock.monotonic() - time < 5 do
        fiber.check_slice()
        -- Long-running operation  --
    end
end
long_operation_fiber = fiber.create(long_operation)
```

The output should look as follows:

```
$ tarantool app.lua
fiber has not yielded for more than 1.500 seconds
FiberSliceIsExceeded: fiber slice is exceeded
```

To learn more about fiber slices, see the *Limit execution time* section.

### Per-module logging

Tarantool 2.11 adds support for modules in the *logging* subsystem. You can configure different log levels for each module using the `box.cfg.log_modules` configuration option. The example below shows how to set the `info` level for `module1` and the `error` level for `module2`:

```
tarantool> box.cfg{log_level = 'warn', log_modules = {module1 = 'info', module2 = 'error
↪'}}
---
...
```

To create a log module, call the `require('log').new()` function:

```
tarantool> module1_log = require('log').new('module1')
---
...
tarantool> module2_log = require('log').new('module2')
---
...
```

Given that `module1_log` has the `info` logging level, calling `module1_log.info` shows a message but `module1_log.debug` is swallowed:

```
tarantool> module1_log.info('Hello from module1!')
2023-05-12 15:10:13.691 [39202] main/103/interactive/module1 I> Hello from module1!
---
...
tarantool> module1_log.debug('Hello from module1!')
---
...
```

Similarly, `module2_log` swallows all events with severities below the `error` level:

```
tarantool> module2_log.info('Hello from module2!')
---
...
```

### HTTP client enhancements

#### Content serialization

The *HTTP client* now automatically serializes the content in a specific format when sending a request based on the specified `Content-Type` header and supports all the Tarantool built-in types. By default, the client uses the `application/json` content type and sends data serialized as JSON:

```
local http_client = require('http.client').new()
local uuid = require('uuid')
local datetime = require('datetime')

response = http_client:post('https://httpbin.org/anything', {
    user_uuid = uuid.new(),
    user_name = "John Smith",
    created_at = datetime.now()
})
```

The body for the request above might look like this:

```
{
    "user_uuid": "70ebc08d-2a9a-4ea7-baac-e9967dd45ac7",
    "user_name": "John Smith",
    "created_at": "2023-05-15T11:18:46.160910+0300"
}
```

To send data in a YAML or MsgPack format, set the `Content-Type` header explicitly to `application/yaml` or `application/msgpack`, for example:

```
response = http_client:post('https://httpbin.org/anything', {
    user_uuid = uuid.new(),
    user_name = "John Smith",
    created_at = datetime.now()
}, {
    headers = {
        ['Content-Type'] = 'application/yaml',
    }
})
```

### Query and form parameters

You can now encode query and form parameters using the new `params` request option. In the example below, the requested URL is `https://httpbin.org/get?page=1`.

```
local http_client = require('http.client').new()

response = http_client:get('https://httpbin.org/get', {
    params = { page = 1 },
})
```

Similarly, you can send form parameters using the `application/x-www-form-urlencoded` type as follows:

```
local http_client = require('http.client').new()

response = http_client:post('https://httpbin.org/anything', nil, {
    params = { user_id = 1, user_name = 'John Smith' },
})
```

### Streaming

The HTTP client now supports chunked writing and reading of request and response data, respectively. The example below shows how to get chunks of a JSON response sequentially instead of waiting for the entire response:

```
local http_client = require('http.client').new()
local json = require('json')

local timeout = 1
local io = http_client:get(url, nil, {chunked = true})
for i = 1, 3 do
     local data = io:read('\r\n', timeout)
     if len(data) == 0 then
         -- End of the response.
         break
     end
     local decoded = json.decode(data)
     -- <..process decoded data..>
end
io:finish(timeout)
```

Streaming can also be useful to upload a large file to a server or to subscribe to changes in `etcd` using the gRPC-JSON gateway. The example below demonstrates communication with the `etcd` stream interface. The request data is written line-by-line, and each line represents an `etcd` command.

```
local http_client = require('http.client').new()

local io = http_client:post('http://localhost:2379/v3/watch', nil, {chunked = true})
io:write('{"create_request":{"key":"Zm9v"}}')
local res = io:read('\n')
print(res)
-- <..you can feed more commands here..>
io:finish()
```

### Linearizable read

Linearizability of read operations implies that if a response for a write request arrived earlier than a read request was made, this read request should return the results of the write request. Tarantool 2.11 introduces the new `linearizable` isolation level for *box.begin()*:

```
box.begin({txn_isolation = 'linearizable', timeout = 10})
box.space.my_space:select({1})
box.commit()
```

When called with `linearizable`, `box.begin()` yields until the instance receives enough data from remote peers to be sure that the transaction is linearizable.

There are several prerequisites for linearizable transactions:

- Linearizable transactions may only perform requests to synchronous, local, or temporary memtx spaces.
- Starting a linearizable transaction requires *box.cfg.memtx_use_mvcc_engine* to be set to `true`.
- The node is the replication source for at least `N - Q + 1` remote replicas. Here `N` is the count of registered nodes in the cluster and `Q` is *replication_synchro_quorum*. So, for example, you can't perform a linearizable transaction on anonymous replicas.

### Explicit sequential scanning in SQL

Tarantool is primarily designed for OLTP workloads. This means that data reads are supposed to be relatively small. However, a suboptimal SQL query can cause a heavy load on a database.

The new `sql_seq_scan` *session setting* is added to explicitly cancel full table scanning. The request below should fail with the `Scanning is not allowed for 'T'` error:

```
SET SESSION "sql_seq_scan" = false;
SELECT a FROM t WHERE a + 1 > 10;
```

To enable table scanning explicitly, use the new `SEQSCAN` keyword:

```
SET SESSION "sql_seq_scan" = false;
SELECT a FROM SEQSCAN t WHERE a + 1 > 10;
```

In future versions, `SEQSCAN` will be required for scanning queries with the ability to disable the check using the `sql_seq_scan` session setting. The new behavior can be enabled using a *corresponding compat option*.

### Strict fencing in RAFT

Leader election is implemented in Tarantool as a modification of the Raft algorithm. The 2.11 release adds the ability to specify the leader fencing mode that affects the leader election process.

---

**Примечание:** Currently, Cartridge does not support leader election using Raft.

---

You can control the fencing mode using the *election_fencing_mode* property, which accepts the following values:

- In `soft` mode, a connection is considered dead if there are no responses for `4 * replication_timeout` seconds both on the current leader and the followers.

- In `strict` mode, a connection is considered dead if there are no responses for `2 * replication_timeout` seconds on the current leader and `4 * replication_timeout` seconds on the followers. This improves the chances that there is only one leader at any time.

### 11.1.4 EOL versions

This section contains information about Tarantool versions that have reached their *end of life* in accordance with the *Tarantool release policy*. This means that these versions don't receive updates and fixes anymore. However, we still provide technical support for certain time after a version's EOL. The current support status is reflected by the **End of support** column of the table below.

For information about major changes between EOL versions, see *Основные изменения*.

**Примечание:** Before 2.10.0, version numbers were subject to the *legacy versioning policy*.

| Version | Release date | End of life | End of support |
|---|---|---|---|
| *2.10.8* | September 14, 2023 | September 14, 2023 | Not planned yet |
| *2.10.7* | May 24, 2023 | September 14, 2023 | Not planned yet |
| *2.10.6* | March 22, 2023 | September 14, 2023 | Not planned yet |
| *2.10.5* | February 20, 2023 | September 14, 2023 | Not planned yet |
| *2.10.4* | November 11, 2022 | September 14, 2023 | Not planned yet |
| *2.10.3* | September 30, 2022 | September 14, 2023 | Not planned yet |
| *2.10.2* | September 1, 2022 | September 14, 2023 | Not planned yet |
| *2.10.1* | August 8, 2022 | September 14, 2023 | Not planned yet |
| *2.10.0* | May 22, 2022 | September 14, 2023 | Not planned yet |
| *2.8.4* | April 25, 2022 | April 25, 2022 | December 31, 2024 |
| *2.8.3* | December 22, 2021 | April 25, 2022 | December 31, 2024 |
| *2.8.2* | August 19, 2021 | April 25, 2022 | December 31, 2024 |
| *2.7.3* | August 19, 2021 | August 19, 2021 | August 19, 2021 |
| *2.7.2* | April 21, 2021 | August 19, 2021 | August 19, 2021 |
| *2.6.3* | April 21, 2021 | April 21, 2021 | April 21, 2021 |
| *2.6.2* | December 30, 2020 | April 21, 2021 | April 21, 2021 |
| *2.5.3* | December 30, 2020 | December 30, 2020 | December 30, 2020 |
| *2.5.2* | October 22, 2020 | December 30, 2020 | December 30, 2020 |
| *2.4.3* | October 22, 2020 | October 22, 2020 | October 22, 2020 |
| *2.4.2* | July 17, 2020 | October 22, 2020 | October 22, 2020 |
| *2.3.3* | July 17, 2020 | July 17, 2020 | July 17, 2020 |
| *2.3.2* | April 20, 2020 | July 17, 2020 | July 17, 2020 |
| *2.2.3* | April 20, 2020 | April 20, 2020 | April 20, 2020 |
| *2.2.2* | December 31, 2019 | April 20, 2020 | April 20, 2020 |
| *1.10.15 LTS* | February 20, 2023 | February 20, 2023 | February, 2024 |
| *1.10.14 LTS* | August 8, 2022 | February 20, 2023 | February, 2024 |
| *1.10.13 LTS* | April 25, 2022 | February 20, 2023 | February, 2024 |
| *1.10.12 LTS* | December 22, 2021 | February 20, 2022 | February 20, 2023 |
| *1.10.11 LTS* | August 19, 2021 | February 20, 2022 | February 20, 2023 |
| *1.10.10 LTS* | April 24, 2021 | February 20, 2022 | February 20, 2023 |
| *1.10.9 LTS* | December 30, 2020 | February 20, 2022 | February 20, 2023 |
| *1.10.8 LTS* | October 22, 2020 | February 20, 2022 | February 20, 2023 |
| *1.10.7 LTS* | July 17, 2020 | February 20, 2022 | February 20, 2023 |
| *1.10.6 LTS* | April 20, 2020 | February 20, 2022 | February 20, 2023 |

continues on next page

Таблица 1 – продолжение с предыдущей страницы

| Version | Release date | End of life | End of support |
|---|---|---|---|
| *1.10.5 LTS* | January 14, 2020 | February 20, 2022 | February 20, 2023 |
| *1.10.4 LTS* | September 26, 2019 | February 20, 2022 | February 20, 2023 |
| *1.10.3 LTS* | April 1, 2019 | February 20, 2022 | February 20, 2023 |
| *1.10.2 LTS* | October 13, 2018 | February 20, 2022 | February 20, 2023 |

### Основные изменения

---

**Важно:** This page is no longer maintained.

The table below lists major changes in Tarantool versions **up to 2.11.0**. For overviews of changes in newer versions, see their *What's new* pages inside *Примечания к версиям*.

---

С каждым выпуском очередной версии Tarantool мы добавляем новые функции и исправления ошибок. Полные списки изменений для каждой версии можно найти в *примечаниях к версиям*. В этой таблице перечислены наиболее значимые изменения с номерами версий, в которых они были добавлены.

To keep track of the major features in Tarantool versions, you can use the table below.

Later versions inherit features from earlier ones in the same release series. Note that versions before 2.10.* are numbered according to the *legacy release policy*, while versions 2.10.0 and later adhere to the *current release policy*. Versions that only include bug fixes are not listed in this table.

| Номер версии | Характеристика |
|---|---|
| *2.11.0* | **Tarantool Enterprise Edition** Security enhancements: *Encrypted SSL/TLS keys*, new *security enforcement options*, *PAP-SHA256 support Read views* `zlib` support for *tuple compression WAL extensions* **Tarantool Community Edition** Pagination support (the *after* option) Support for downgrading a database (*box.schema.downgrade()*) *New bootstrap strategy Limitation of fiber execution time* Per-module logging HTTP client enhancements Linearizable read Explicit sequential scanning in SQL *Strict fencing in RAFT* |
| 2.10.5 | Introduced the `_vspace_sequence` system space view of the `_space_sequence` system space (gh-7858). The log produced during `box.cfg{}` now contains the build target triplet (for example, `Linux-x86_64-RelWithDebInfo`). |
| 2.10.4 | The JSON log format can now be used with the syslog logger (gh-7860). SQL improvements: `CASE` (gh-6990) and `NULLIF()` (gh-6989). Diagnostics now provide relative file paths instead of absolute ones (gh-7808). |
| 2.10.3 | RedOS 7.3 is now supported. Added the `-DENABLE_HARDENING=ON/OFF` CMake option that enables hardening against memory corruption attacks (gh-7536). |
| 2.10.2 | Internal fibers cannot be cancelled from the Lua public API anymore (gh-7473) |
| 2.10.1 | Interactive transactions are now possible in remote binary consoles (gh-7413) Improved string representation of datetime intervals (gh-7045) |
| 2.10.0 | Transaction isolation levels in Lua and IPROTO (gh-6930) Fencing and pre-voting in RAFT (gh-6661) *Foreign keys* and *constraints* support (gh-6436) *New DATETIME type* HTTP/2 support for the HTTP client Preliminary support for GNU/Linux ARM64 and MacOS M1 (gh-2712, gh-6065, gh-6066, gh-6084, gh-6093, gh-6098, gh-6189) *Streams and interactive transactions in iproto* (gh-5860) *Consistent SQL type system* Faster `net.box` module performance (improved up to 70%) (gh-6241) Compact mode for tuples (gh-5385) `memtx_allocator` option in `box.cfg{}` (gh-5419) |
| 2.8.2 | Symbolic log levels in the `log` module (gh-5882) |
| 2.7.3, 1.10.11 | `LJ_DUALNUM` mode support in `luajit-gdb` (gh-6224) |
| 2.7.3 | New `table.equals` method in Lua `box.info.synchro` interface for synchronous replication statistics (gh-5191) |
| 2.8.1 | *Multiple iproto threads* (gh-5645) Set *box.cfg* options with environment variables (gh-5602) Friendly *LuaJIT memory profiler report* (gh-5811) `--leak-only` LuaJIT memory profiler option (gh-5812) |
| 2.7.1 | *LuaJIT memory profiler* (gh-5442) SQL *ALTER TABLE ADD COLUMN* statement support for empty tables (gh-2349, gh-3075) |
| 2.6.3, 2.7.2 | Очередь упреждающей журнализации (gh-5536) |
| 2.6.3, 2.7.2, 2.8.1 | Функция *box.ctl.promote()* и финализация выборов вручную (gh-3055) |
| 2.6.1 | *LuaJIT platform metrics* (gh-5187) *Automated leader election* based on Raft algorithm (gh-1146) *Transactional manager* for memtx engine (gh-4897) |
| 2.5.3, 2.6.2, 2.7.1 | Возможность задать *replication_synchro_quorum* в виде символьного выражения (gh-5446) |
| 2.5.3, 2.6.2 | Функция *box.ctl.is_recovery_finished()* для движка memtx (gh-5187) |
| 2.5.1 | *Synchronous replication* (beta) (gh-4842) Allow an *anonymous replica* to follow another anonymous replica (gh-4696) |
| 2.4.1 | *UUID type for field and index* (gh-4268, gh-2916) *popen* built-in module (gh-4031) Ability to create *custom error types* (gh-4398) *Transparent marshalling* through `net.box` (gh-4398) *Stacked diagnostic area* (gh-1148) |
| 2.3.1 | *Field name and JSON path updates* (gh-1261) *Anonymous replica* type (gh-3186) *DOUBLE* type in SQL (gh-3812) Support for *decimals* in spaces, `decimal` field type (gh-4333) *fiber.top()* function in Lua (gh-2694) Feed data from memory during replica initial join (gh-1271) SQL prepared statements support and cache (gh-2592, gh-3292) *_session_settings* service space (gh-4511) |

### Tarantool legacy release policy

**Важно:** This page describes the release policy that was used for Tarantool versions **before 2.10.0**.

For information about the current release policy, see *Tarantool release policy*.

Версия Tarantool состоит из трех чисел, например 2.6.2 или 1.10.9:

- Первое число обозначает серию ОСНОВНЫХ версий, в которой вводятся некоторые *значительные изменения*. Пока был только один переход на новую серию, когда мы выпустили серию 2.x с поддержкой SQL.
- Второе число обозначает серию ПРОМЕЖУТОЧНЫХ версий, которые используются для введения новых *функций*. Эти версии могут быть *обратно несовместимы* между собой.
- Третье число используется для обозначения ПАТЧ-версий, которые отражают стабильность ПРОМЕЖУТОЧНЫХ версий:
  - `0` — **альфа**
  - `1` — **бета**
  - `2` и выше — **релизная версия** (раньше называлась **стабильная**).

Таким образом, разработка каждой ПРОМЕЖУТОЧНОЙ версии проходит жизненный цикл следующим образом:

1. **Альфа-версия**. Раз в квартал мы начинаем разработку новой альфа-версии, например 2.3.0, 2.4.0 и так далее. Это не совсем альфа-версия, как в типичном жизненном цикле выпуска программного обеспечения, а скорее текущая основная версия, которая находится в процессе интенсивной разработки и может быть нестабильной. Текущая альфа-версия всегда живет в основной ветви разработки.
2. **Бета-версия**. Когда все запланированные функции реализованы, мы отделяем новую ветвь от основной и помечаем ее как новую бета-версию с `1` в конце для обозначения ПАТЧА, например, 2.3.1, 2.4.1 и так далее. Эту версию пока нельзя назвать стабильной (только что перестали добавлять функциональные возможности), хотя с момента последнего стабильного выпуска в ней нет известных критических регрессионных ошибок.
3. **Релизная версия** (раньше называлась **стабильная**). Когда мы видим, что бета-версия успешно работает в течение еще одного квартала в рабочей среде или в среде разработки, пока мы исправляем обнаруженные ошибки, мы объявляем эту версию стабильной. ПАТЧ помечается цифрой 2, например, 2.3.2, 2.4.2 и так далее.

   Мы поддерживаем такую версию в течение 3 месяцев, параллельно работая над еще одной стабильной версией, в которой исправляем все найденные ошибки. Через три месяца мы выпускаем ее, обозначая ПАТЧ цифрой `3`, например, 2.3.3, 2.4.3 и так далее. После установки этого тега никакие новые изменения в релизную ветвь не вносим, объявляем ее устаревшей и заменяем ее новой ПРОМЕЖУТОЧНОЙ версией.

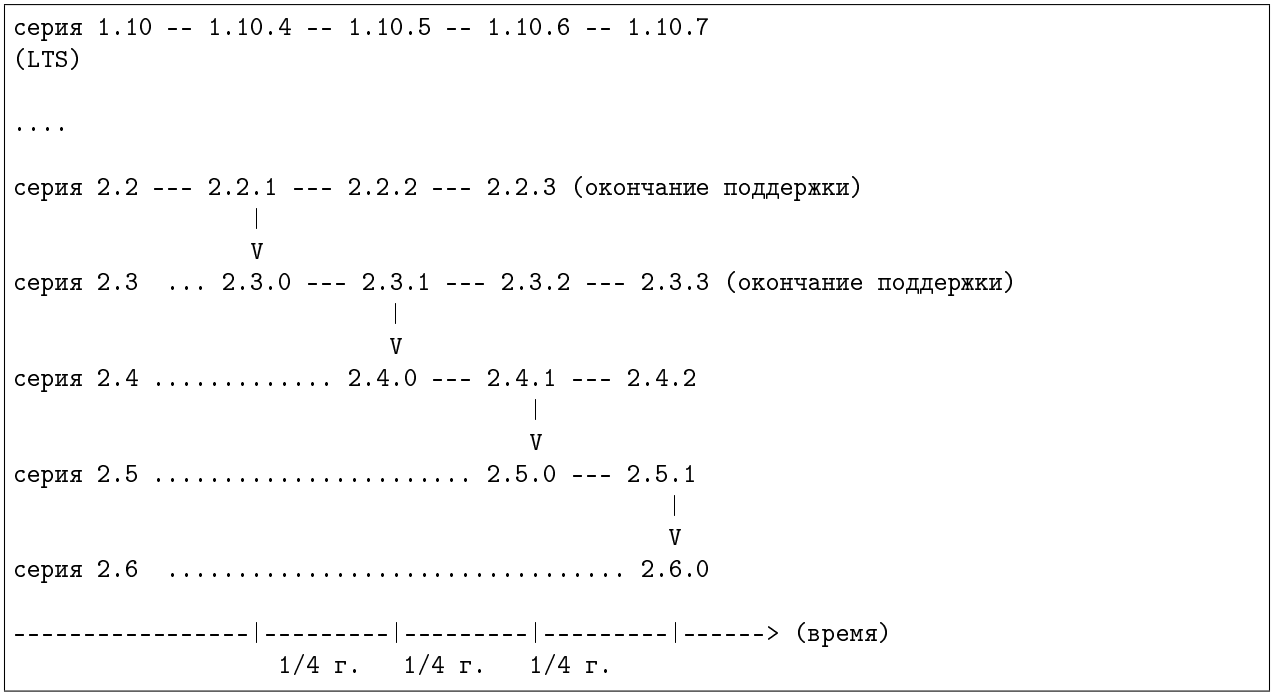
   В релизные версии мы не добавляем никакие новые функциональные возможности, а только обратно совместимые исправления.

Как в Ubuntu, у нас есть две категории стабильных версий в плане поддержки:

- **LTS-версии (долгосрочная поддержка, long term support)** — это выпуски, которые поддерживаются 3 года для сообщества и 5 лет для платящих клиентов. Текущая серия LTS-версий — 1.10, для нее выпускаются только версии уровня ПАТЧ.

- **Обычные версии** — это серия выпусков, которая поддерживается только в течение нескольких месяцев до выхода следующей стабильной серии.

Ниже представлена схема, которая иллюстрирует вышеописанную последовательность выпуска версий на примере некоторых последних версий и серий:




*Поддержка* означает, что мы продолжаем исправлять ошибки. Мы также добавляем исправления ошибок в следующие серии версий: LTS, последнюю стабильную, бета и альфа. Если посмотреть на схему выпуска версий выше, — это означает, что исправления ошибок будут добавлены в серии 1.10, 2.4, 2.5 и 2.6.

Итак, раз в квартал выходят (см. вышеприведенную схему выпуска версий):

- очередная LTS-версия, например 1.10.9
- две стабильные версии, например 2.5.3 и 2.6.2
- бета-версия следующей серии, например 2.7.1.

Когда мы находим и исправляем известную уязвимость (CVE) в любой поддерживаемой версии, мы выпускаем для этого патч, но не проставляем тег уровня ПАТЧ. Пользователи узнают о таких критических патчах из официального новостного канала Tarantool (tarantool_news).

Мы также публикуем ночные сборки и используем четвертый слот в идентификаторе версии для обозначения номера ночной сборки.

---

**Примечание:** В новой серии релизов могут быть обратно несовместимые изменения — это значит, что код на Lua, SQL или C, выполняемый в текущей серии версий, может не выполняться в следующей серии версий. Тем не менее, мы не злоупотребляем этим правилом и не вносим несовместимые изменения без причины. Обычно мы предоставляем информацию о том, насколько сформирована функциональность, в примечаниях к версии.

Обратите внимание, что структура бинарных данных всегда совместима с предыдущими сериями, а также с сериями LTS (экземпляр версии `X.Y` может быть запущен на основе данных `X.(Y+1)` или `1.10.z`). Бинарный протокол также обратно совместим (как клиент-серверный протокол, так и протокол

репликации).

### Tarantool 2.10.8

Released on 2023-09-14.

- Release: v. 2.10.8.
- Tag: `2.10.8`.

#### Overview

2.10.8 is the ninth *stable* version of the 2.10 release series. It introduces 5 improvements and resolves 28 bugs since 2.10.7.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

#### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

#### Functionality added or changed

##### Box

- The maximum length of `box.cfg{}` string parameters is now 512 instead of 256.

##### Test

##### Fuzz

- LuaJIT now can be fuzzed using grammar-based fuzzer (gh-4823).

##### Build

- Hardening against memory corruption attacks is now enabled by default on FreeBSD (gh-7536).
- Added the CMake option `FIBER_STACK_SIZE` to set the default fiber stack size.
- Updated libcurl to version 8.3.0.

### Bugs fixed

#### Core

- Fixed a bug when Tarantool failed to decode a request containing an unknown IPROTO key. The bug resulted in broken connectivity between Tarantool 2.10 and 2.11 (gh-8745).
- Fixed a bug causing the `ER_CURSOR_NO_TRANSACTION` failure for transactions on synchronous spaces when the `on_commit/on_rollback` triggers are set (gh-8505).
- Fixed a bug causing the effective session and user are not propagated to `box.on_commit` and `box.on_rollback` trigger callbacks when transaction is synchronous (gh-8742).
- Fixed a crash that could happen when Tarantool is started in the *background mode* (gh-6128).
- Fixed a bug when MVCC sometimes lost gap record (gh-8326).
- Fixed a bug when MVCC rollback of prepared statement could break internal invariants (gh-8648).
- Now MVCC engine automatically aborts a transaction if it reads changes of a prepared transaction and this transaction is aborted (gh-8654).
- Fixed a bug that caused writing incorrect values into the `stream_id` field of xlog headers (gh-8783).
- Fixed a bug when a space that is referenced by a foreign key could not be truncated even if the referring space was empty (gh-8946).
- Fixed a crash that could happen when Tarantool is compiled by `clang` version 15 and above with enabled AddressSanitizer (tarantool/tarantool-qa#321).
- Fixed a use-after-free bug in iproto server code (gh-9037).
- Fixed a heap-buffer-overflow bug in fiber creation code (gh-9026).

#### Memtx

- Fixed a heap-use-after-free bug in the transaction manager, which could occur when performing a DDL operation concurrently with a transaction on the same space (gh-8781).

#### Vinyl

- Fixed a heap-use-after-free bug in the Vinyl read iterator caused by a race between a disk read and a memory dump task. The bug could lead to a crash or an invalid query result (gh-8852).

#### Replication

- Fixed a possible failure to promote the desired node by `box.ctl.promote()` on a cluster with nodes configured with `election_mode = "candidate"` (gh-8497).
- Fixed nodes configured with `election_mode = 'candidate'` spuriously detecting a split-vote when another candidate should win with exactly a quorum of votes for it (gh-8698).

### LuaJIT

Backported patches from the vanilla LuaJIT trunk (gh-8516, gh-8825). The following issues were fixed as part of this activity:

- Fixed canonicalization of +-0.0 keys for `IR_NEWREF`.
- Fixed result truncation for `bit.rol` on x86_64 platforms.
- Fixed `lua_yield()` invocation inside C hooks.
- Fixed memory chunk allocation beyond the memory limit.
- Fixed TNEW load forwarding with instable types.
- Fixed use-def analysis for `BC_VARG`, `BC_FUNCV`.
- Fixed `BC_UCLO` insertion for returns.
- Fixed recording of `BC_VARG` with unused vararg values.
- Initialization instructions on trace are now emitted only for the first member of a union.
- Prevent integer overflow while parsing long strings.
- Fixed various `^` operator and `math.pow()` function inconsistencies.
- Fixed parsing with predicting `next()` and `pairs()`.
- Fixed binary number literal parsing. Parsing of binary number with a zero fractional part raises error too now.
- Fixed load forwarding optimization applied after table rehashing.
- Fixed recording of the `BC_TSETM`.

### Lua

- Fixed the xlog reader Lua module to show unknown row header fields. Before this change the xlog reader silently skipped them.

### Netbox

- Fixed a heap-use-after-free bug in the function creating a tuple format Lua object for `net.box` (gh-8889).

### Box

- Fixed the memory leaks caused by the multi-statement transaction errors in the space index building and the space format checking operations (gh-8773).
- Fixed a bug in the box console implementation because of which the `language` parameter was shared between connected clients (gh-8817).
- Fixed an invalid memory access in a corner case of a specialized comparison function (gh-8899).

### Console

- Fixed console ignoring `-i` flag in case stdin is not a tty (gh-5064).

### Datetime

- Fixed a bug raising a false positive error when creating new intervals with range boundary values (gh-8878).
- Fixed a bug with buffer overflow in `tnt_strptime` (gh-8502).

### Http

- Fixed a streaming connection stuck if etcd is stopped unexpectedly (gh-9086).

### Msgpack

- Fixed decoding datetime intervals with fields larger than possible int32 values (gh-8887).

### Tarantool 2.10.7

Released on 2023-05-24.

- Release: v. 2.10.7.
- Tag: `2.10.7`.

### Overview

2.10.7 is the 8th *stable* version of the 2.10 release series. It resolves 17 bugs since 2.10.6.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Bugs fixed

#### Core

- Fixed a crash that could happen when preparing a crash report on macOS (gh-8445).
- Fixed an integer overflow issue in `net.box` (ghs-121).
- An `IPROTO_EVENT` packet now has the same sync number as the last corresponding `IPROTO_WATCH` request (gh-8393).
- Fixed a bug because of which a dirty (not committed to WAL) DDL record could be written to a snapshot and cause a recovery failure (gh-8530).

#### Replication

- Fixed a bug that occurred on applier failure: a node could start an election without having a quorum to do this (gh-8433).
- Now if a join fails with some non-critical error, such as `ER_READONLY`, `ER_ACCESS_DENIED`, or something network-related, the instance tries to find a new master to join off and tries again (gh-6126, gh-8681).
- States when joining a replica are renamed. Now the value of `box.info.replication[id].upstream.status` during join can be either `wait_snapshot` or `fetch_snapshot` instead of `initial_join` (gh-6126).
- Fixed replicaset bootstrap getting stuck on some nodes with `ER_READONLY` when there are connectivity problems (gh-7737, gh-8681).
- Fixed a bug when a replicaset state machine that is tracking the number of appliers according to their states could become inconsistent during reconfiguration (gh-7590).

#### LuaJIT

Backported patches from the vanilla LuaJIT trunk (gh-8069, gh-8516). The following issues were fixed as part of this activity:

- Fixed `emit_rma()` for x64/GC64 mode for non-`mov` instructions.
- Limited Lua C library path with the default `PATH_MAX` value of 4096 bytes.
- Fixed assembling of `IR_LREF` assembling for GC64 mode on x86_64.

#### SQL

- Fixed an assertion when selecting tuples with incomplete internal format (gh-8418).
- Fixed integer overflow issues in built-in functions (ghs-119).
- Fixed a possible assertion or segmentation fault when optimizing `INSERT INTO ... SELECT FROM` (gh-8661).
- Fixed an integer overflow issue and added check for the `printf()` failure due to too large size (ghs-122).

### Datetime

- Fixed an error in `datetime.set` when `timestamp` is passed along with `nsec`, `usec`, or `msec` (gh-8583).
- Fixed errors when the string representation of a datetime object had a negative nanosecond part (gh-8570).

### Build

- Enabled compiler optimizations for static build dependencies, which were erroneously disabled in version 2.10.3 (gh-8606).

## Tarantool 2.10.6

Released on 2023-03-22.

- Release: v. 2.10.6.
- Tag: `2.10.6`.

### Overview

2.10.6 is the 7th *stable* version of the 2.10 release series. It resolves 3 bugs since 2.10.5.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Bugs fixed

### Core

- Fixed various bugs related to unsafe (i.e., coming from an unknown source) decoding and validating of MsgPack extensions (ghs-73).

### LuaJIT

Backported patches from the vanilla LuaJIT trunk (gh-8069). The following issues were fixed as part of this activity:

- Fixed successful `math.min/math.max` call with no args (gh-6163).
- Fixed inconsistencies in `math.min/math.max` calls with a NaN arg (gh-6163).
- Fixed `pcall()` call without arguments on arm64.
- Fixed assembling of `IR_{AHUV}LOAD` specialized to boolean for aarch64.
- Fixed constant rematerialization on arm64.

### Box

- Fixed a bug where `box.cfg.force_recovery` doesn't work when there is no user spaces in a snapshot (gh-7974).

## Tarantool 2.10.5

Released on 2023-02-20.

- Release: v. 2.10.5.
- Tag: `2.10.5`.

### Overview

2.10.5 is the sixth *stable* version of the 2.10 release series. It introduces 5 improvements and resolves 44 bugs since 2.10.4.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Functionality added or changed

### Core

- Introduced the `_vspace_sequence` system space view of the `_space_sequence` system space (gh-7858).
- The log produced during `box.cfg{}` now contains the build target triplet (for example, `Linux-x86_64-RelWithDebInfo`).

### Build

- OpenSUSE 15.1 and 15.2 are no longer supported.
- Updated libcurl to version 7.87.0 (gh-8150).
- Alpine Linux 3.16 is now supported.

### Bugs fixed

### Core

- Fixed a bug in fiber switching that could lead to a segmentation fault error on AArch64 systems (gh-7523, gh-7985).
- Fixed wrong CPU architecture reported in `tarantool.build.target` on M1/M2 Macs (gh-7495).
- Fixed a bug when fields could be removed from a table stored in a variable when a logging function was called on this variable (for example, `log.info(a)`) (gh-3853).
- Fixed a logging bug: when logging tables with fields that have reserved internal names (such as `pid`) in the plain log format, such fields weren't logged (gh-3853).
- Added the `message` field when logging tables without such field in the JSON log format (gh-3853).
- Fixed an assertion on malformed JSON message written to the log (gh-7955).
- Fixed the bug because of which `box.session.on_auth` triggers were not invoked if the authenticated user didn't exist (gh-8017).
- Eliminated the possibility of user enumeration by analyzing errors sent in reply to malformed authentication requests (ghs-21).
- Fixed a bug when Tarantool could execute random bytes as a Lua code after fork on systems with a glibc version earlier than 2.29 (gh-7886).
- A referenced space or a function being used in a constraint can now be dropped in the same transaction with the referencing constraint or space (gh-7339).
- Fixed Tarantool being stuck during a crash on macOS (gh-8023).
- Fixed a bug that prevented collection of crash reports (gh-8083).
- Fixed a crash in `net.box` that happened if the error message raised by the server contained `printf` formatting specifiers, such as `%d` or `%s` (gh-8043).
- Fixed read-only statements executing successfully in transactions that were aborted by yield or timeout. Now, read-only statements fail in this case, just like write statements (gh-8123).
- Fixed a transaction conflict reported mistakenly when a key was deleted twice with MVCC engine enabled (gh-8122).
- `net.box` connections now contain information about sequences used by remote spaces (gh-7858).
- Fixed a crash that happened if a transaction was aborted (for example, by fiber yield with MVCC off) while the space's `on_replace` or `before_replace` trigger was running (gh-8027).
- Fixed a possible crash when attempting to update the same field in `tuple/space/index:update()` more than once (gh-8216).
- Fixed empty BITSET indexes crashing on `len` calls (gh-5809).
- Fixed a crash when functional indexes were used with very specific chunk size (gh-6786).

### Memtx

- Fixed a possible repeatable read violation with reverse iterators (gh-7755).
- Fixed a crash on series of transactions in memtx (gh-7756).
- Fixed a phantom read that could happen after reads from different indexes followed by a rollback (gh-7828).
- Fixed an assert in the MVCC engine (gh-7945).
- Fixed an assertion failure in MVCC during statement preparation (gh-8104).
- Fixed possible loss of a committed tuple after rollback of a prepared transaction (gh-7930).

### Vinyl

- Fixed a bug that could result in `select()` skipping an existing tuple after a rolled back `delete()` (gh-7947).

### Replication

- Fixed local space writes failing with error `Found uncommitted sync transactions from other instance with id 1` when synchronous transaction queue belongs to another instance and isn't empty (gh-7592).
- Fixed an assertion failure on master when a replica resubscribes with a smaller vclock than previously seen (gh-5158).
- A warning is now raised when `replica_id` is changed by a `before_replace` trigger while adding a new replica. Previously, there was an assertion checking this (gh-7846).
- Fixed a segmentation fault that happened when a `before_replace` trigger set on space `_cluster` returned nil (gh-7846).
- Fixed possible transaction conflict errors on applying a replication stream (gh-8121).

### Raft

- Fixed an assertion failure in case when an election candidate is reconfigured to a voter during an ongoing WAL write (gh-8169).
- Fixed nodes configured with `election_mode = "manual"` sometimes increasing the election term excessively after their promotion (gh-8168).

### LuaJIT

Backported patches from vanilla LuaJIT trunk (gh-7230). In the scope of this activity, the following issues have been resolved:

- Fix `io.close()` for already closed standard output.
- Fix trace execution and stitching inside vmevent handler (gh-6782).
- Fixed `emit_loadi()` on x86/x64 emitting xor between condition check and jump instructions.
- Fix stack top for error message when raising the OOM error (gh-3840).

- Enabled external unwinding on several LuaJIT platforms. Now it is possible to handle ABI exceptions from Lua code (gh-6096).
- Disabled math.modf compilation due to its rare usage and difficulties with proper implementation of the corresponding JIT machinery.
- Fixed inconsistent behaviour on signed zeros for JIT-compiled unary minus (gh-6976).
- Fixed `IR_HREF` hash calculations for non-string GC objects for GC64.
- Fixed assembling of type-check-only variant of `IR_SLOAD`.
- Enabled the platform profiler for Tarantool built with GC64 mode (gh-7919).
- Added full-range lightuserdata support to the `luajit-gdb.py` extension (gh-6481).

Backported patches from vanilla LuaJIT trunk (gh-8069). In the scope of this activity, the following issues have been resolved:

- Fixed loop realigment for dual-number mode
- Fixed os.date() for wider libc strftime() compatibility.
- Fix interval parsing for sysprof for dual-number mode.

### Lua

- Fixed alias detection in the YAML serializer in case the input contains objects that implement the `__serialize` meta method (gh-8240).

### SQL

- Fixed a bug when collation could change the type of a built-in function argument (gh-7992).
- Fixed several bugs happening because of improper handling of `malloc()` failures (ghs-65, ghs-66, ghs-67, ghs-68).

### Box

- Fixed a possible error during rollback of read-only transaction statements (gh-5501).
- Fixed a bug in `space_object:create_index()` when `collation` option is not set. Now it is inherited from the space format (gh-5104).
- Eliminated a code injection vulnerability in the processing of the `replication_synchro_quorum` `box.cfg()` option (ghs-20, GHSA-74jr-2fq7-vp42).

### Datetime

- Fixed a segmentation fault that happened when the value passed to the `%f` modifier of `datetime_object:format()` was too big (ghs-31).

### Fiber

- Fixed the assertion fail in `cord_on_yield` (gh-6647).

### Log

- Fixed an incorrect facility value in syslog on Alpine and OpenBSD (gh-8269).

### Build

- Fixed `-Werror` build fail on Clang 15 (gh-8110).

## Tarantool 2.10.4

Released on 2022-11-11.

- Release: v. 2.10.4.
- Tag: `2.10.4`.

### Overview

2.10.4 is the fifth *stable* version of the 2.10 release series. It introduces 5 improvements and resolves 28 bugs since 2.10.3.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

---

**Примечание:** Now the empty string, `n`, `nu`, `s`, and `st` (that is, leading parts of `num` and `str`) are not accepted as valid field types (gh-5940). *This instruction* will help you upgrade to Tarantool 2.10.4 and newer if you've previously used these values in field types.

---

### Functionality added or changed

#### Core

- The JSON log format can now be used with the syslog logger (gh-7860).

#### SQL

- New rules are applied to determine the type of CASE operation (gh-6990).
- Now `NULLIF()` call results have the same type as its first argument (gh-6989).

#### Build

- Diagnostics now provide relative file paths instead of absolute ones (gh-7808).
- Now the compiler info displayed in `tarantool.build.compiler` and `tarantool --version` shows the ID and the version of the compiler that was used to build Tarantool. The output has the format `${CMAKE_C_COMPILER_ID}-${CMAKE_C_COMPILER_VERSION}`, for example, `Clang-14.0.0.14000029` (gh-7888).

### Bugs fixed

#### Core

- Fixed creation of spaces with a constraint and a foreign key on the same field (gh-7645).
- Now the same error is returned when a password or a username provided during authorization is incorrect. This prevents user enumeration (ghs-16).
- Added boundary checking for `getenv()` return values. Also, for security reasons, Tarantool code now copies these values instead of using them directly (gh-7797).
- `os.getenv()` now always returns values of sane size (gh-7797).
- Fixed the `BEGIN`, `COMMIT`, and `ROLLBACK` counters in the `box.stat()` output. Now they show the number of started, committed, and rolled back transactions (gh-7583).
- Fixed a crash that could occur during log rotation and application exit (gh-4450).
- Fixed a possible buffer overflow in `mp_decode_decimal()` and `decimal_unpack()` when an input string was too long (ghs-17).
- Fixed a bug in the MsgPack library that could lead to a failure to detect invalid MsgPack input and, as a result, an out-of-bounds read (ghs-18).
- If an error occurs during a snapshot recovery, its log now contains information about the row that caused the error (gh-7917).

### Memtx

- Fixed possible loss of committed tuples in secondary indexes with MVCC transaction manager (gh-7712).
- Fixed an assertion being triggered on `space:drop` (gh-7757).
- Fixed possible violation of the secondary index uniqueness with the transaction manager enabled (gh-7761).

### LuaJIT

- Backported patches from vanilla LuaJIT trunk (gh-7230). In the scope of this activity, the following issues have been resolved:
  - Fix overflow check in `unpack()` optimized by a compiler.
  - Fix recording of `tonumber()` with cdata argument for failed conversions (gh-7655).
  - Fix concatenation operation on cdata. It always raises an error now.
- Fixed the Lua stack dump command (`lj-stack`) to support Python 2: unpacking arguments within the list initialization is not supported in it (gh-7458).

### Lua

- Fixed a crash in `msgpack.decode` in case the input string contains an invalid MsgPack header `0xc1` (gh-7818).

### SQL

- Fixed an assertion when `INDEXED BY` was used with an index that was at least third in a space (gh-5976).
- Fixed a crash that could occur when selecting tuples with more fields than specified in the space format (gh-5310, gh-4666).
- Fixed an assertion in `JOIN` when using an unsupported index (gh-5678).
- Creating indexes on newly added fields no longer leads to assertions in `SELECT` queries (gh-5183).
- Re-running a prepared statement that generates new auto-increment IDs no longer causes an error (gh-6422).
- An error is now thrown if too many indexes were created in SQL (gh-5526).

### Box

- Revoked execute access rights to the `LUA` function from the public role (ghs-14).
- Now the empty string, `n`, `nu`, `s`, and `st` (that is, leading parts of `num` and `str`) are not accepted as valid field types (gh-5940). *This instruction* will help you upgrade to Tarantool 2.10.4 and newer if you've previously used these values in field types.
- Fixed a bug when `type = box.NULL` in `key_def.new()` resulted in `type = 'unsigned'` (gh-5222).
- The `_vfunc` system space now has the same format as `_func` (gh-7822).
- Fixed a crash on recovery from snapshots that don't include system spaces (gh-7800).
- Fixed a bug that occurred when a foreign key was created together with fields that participate in that foreign key (gh-7652).

### Datetime

- Fixed interval arithmetic for boundaries crossing DST (gh-7700).

  Results of datetime arithmetic operations could get a different timezone if the DST boundary has been crossed during the operation:

```
tarantool> datetime.new{year=2008, month=1, day=1,
                        tz='Europe/Moscow'} +
           datetime.interval.new{month=6}
---
- 2008-07-01T01:00:00 Europe/Moscow
...
```

  Now we resolve `tzoffset` at the end of operation if `tzindex` is not 0.

- Fixed subtractions for datetimes with different timezones (gh-7698).

  Previously, the timezone difference (`tzoffset`) was ignored in datetime subtraction operations:

```
tarantool> datetime.new{tz='MSK'} - datetime.new{tz='UTC'}
---
- +0 seconds
...
tarantool> datetime.new{tz='MSK'}.timestamp -
           datetime.new{tz='UTC'}.timestamp
---
- -10800
...
```

  Now this difference is accumulated in the minute component of the resulting interval:

```
tarantool> datetime.new{tz='MSK'} - datetime.new{tz='UTC'}
---
- -180 minutes
...
```

### Tarantool 2.10.3

Released on 2022-09-30.

- Release: v. 2.10.3.
- Tag: `2.10.3`.

#### Overview

2.10.3 is the fourth *stable* version of the 2.10 release series. It introduces 2 improvements and resolves 19 bugs since 2.10.2.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

#### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

#### Functionality added or changed

##### Build

- RedOS 7.3 is now supported.
- Added the `-DENABLE_HARDENING=ON/OFF` CMake option that enables hardening against memory corruption attacks (gh-7536).

#### Bugs fixed

##### Core

- Fixed a bug introduced in Tarantool 2.10.2: log messages could be written to data files thus causing data corruption. The issue was fixed by reverting the fix for gh-4450.
- Switched from MT-Unsafe `strerror()` to MT-Safe `strerror_r()`. The usage of the unsafe function could result in corrupted error messages.
- Fixed a bug when a single JSON update couldn't insert and update a field of a map or an array in two sequential calls. It would either crash or return an error (gh-7705).

#### Memtx

- Fixed incorrect handling of transaction conflicts in full scans by HASH indexes (gh-7493).
- Fixed `use after free` that could occur in the transaction manager in certain states (gh-7449).
- Fixed possible phantom reads with `get` on TREE indexes containing nullable parts (gh-7685).
- Fixed an inconsistency in `index:random` in the context of transaction management (gh-7670).
- Fixed unserializable reads tracked incorrectly after transaction rollbacks (gh-7343).

#### Replication

- Fixed a bug when a fiber committing a synchronous transaction could hang if the instance got a term bump during that or its synchro-queue was fenced in any other way (gh-7253).
- Fixed master occasionally deleting xlogs needed by replicas even without a restart (gh-7584).

#### Raft

- Fixed a bug when `box.ctl.promote()` could hang and bump thousands of terms in a row if called on more than one node at the same time (part of gh-7253).
- Fixed a bug when a node with `election_mode='voter'` could hang in `box.ctl.promote()` or become a leader (part of gh-7253).
- Fixed a bug when a replicaset could be split into parts if a node voted for another instance while having local WAL writes unfinished (part of gh-7253).

#### Lua

#### Merger

- Fixed `use after free` that could occur during iteration over `merge_source:pairs()` or `merger:pairs()` (gh-7657).

#### Popen

- Fixed a race condition in `<popen handle>:signal()` on Mac OS 12 and newer (gh-7658).

#### Box

- Fixed a bug when `fiber.yield()` might break the execution of a shutdown trigger (gh-7434).
- Fixed a possible high CPU usage caused by shutdown triggers (gh-6801).

### Synchro

- Fixed assertions in debug builds and undefined behaviour in release builds when simultaneous elections started or another instance was promoted while an instance was acquiring or releasing the synchro queue (gh-7086).

### Uri

- Fixed a bug in the URI parser: tarantoolctl could not connect when the host name was skipped (gh-7479).

## Tarantool 2.10.2

Released on 2022-09-01.

- Release: v. 2.10.2.
- Tag: `2.10.2`.

### Overview

2.10.2 is the third *stable* version of the 2.10 release series. It introduces 1 improvement and resolves 8 bugs since 2.10.1.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Functionality added or changed

### Core

- Certain internal fibers, such as the connection's worker fiber, vinyl fibers, and some other fibers, cannot be cancelled from the Lua public API anymore (gh-7473).

### Bugs fixed

#### Core

- Fixed a crash of secondary indexes without hints (a critical regression found in 2.10.1) (gh-7605).
- Fixed a possible crash on concurrent `fiber_object:join()` (gh-7489).
- Fixed a potential nil dereference and a crash in case of an active log rotation during the program exit stage (gh-4450).
- Fixed crashes and undefined behaviour of triggers clearing other triggers (gh-4264).

#### Replication

- Fixed `box.info.replication[id].downstream.lag` growing indefinitely on a server when it's not writing any new transactions (gh-7581).

#### Box

- Fixed multiline commands being saved to `~/.tarantool_history` as separate lines (gh-7320).
- Fixed inheritance of field options in indexes when index parts are specified the old Tarantool 1.6 style: `{<field>, <type>, ...}` (gh-7614).
- Fixed unauthorized inserts into the `_truncate` space (ghs-5).

### Tarantool 2.10.1

Released on 2022-08-08.

- Release: v. 2.10.1.
- Tag: `2.10.1`.

#### Overview

**Предупреждение:** It is highly recommended to use Tarantool *v. 2.10.2* instead.

The 2.10.1 release introduced a severe regression (gh-7605), which may pass testing with a low amount of data but impact a production server heavily. It may crash the process and, that is worse, feed incorrect data. The Tarantool development team has decided to remove all the packages associated with this release.

2.10.1 is the second *stable* version of the 2.10 release series. It introduces 17 improvements and resolves 52 bugs since 2.10.0.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

Notable changes are:

- Interactive transactions are now possible in remote binary consoles.
- Improved the string representation of datetime intervals.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Functionality added or changed

#### Core

- **[Breaking change]** Conflicted transactions now throw the `Transaction has been aborted by conflict` error on any CRUD operations until they are either rolled back (which will return no error) or committed (which will return the same error) (gh-7240).
- Read-view transactions now become conflicted on attempts to perform DML statements immediately. Previously, this was detected only on the transaction preparation stage, that is, when calling `box.commit` (gh-7240).
- Interactive transactions are now possible in remote binary consoles (gh-7413).
- It is now possible to omit `space` in declarations of foreign keys that refer to the same space (gh-7200).

#### Datetime

- Improved the string representation of datetime intervals. Now nanoseconds aren't converted and attached to seconds; the intervals are displayed "as is". Example:

```
local ival = date.interval.new{year = 12345, hour = 48, min = 3, sec = 1,
                               nsec = 12345678}
print(tostring(ival))
```

Before:

```
'+12345 years, 48 hours, 3 minutes, 1.012345678 seconds'
```

Now:

```
'+12345 years, 48 hours, 3 minutes, 1 seconds, 12345678 nanoseconds'
```

#### Decimal

- Added C module API for decimals (gh-7228).
- Added Lua/C accessors for decimals into the module API (gh-7228).

#### Tuple

- Added the `box_tuple_field_by_path()` function into the module API. It allow the access to tuple fields from C code using a JSON path (gh-7228).

#### Build

- Fedora 30, 31, 32, and 33 are no longer supported.
- Ubuntu 20.10 (Groovy Gorilla) and 21.04 (Hirsute Hippo) are no longer supported.
- Updated libcurl to version 7.84.0.
- Updated OpenSSL used for static builds to version 1.1.1q.
- Ubuntu 21.10 (Impish Indri) is no longer supported.
- Updated Ncurses used for static builds to version 6.3-20220716 .
- Updated Readline used for static builds to version 8.0p1.
- Updated libyaml to the version with fixed stack overflows.
- Updated zstd to version 1.5.2.
- Updated zlib used for static builds to version 1.2.12.

### Bugs fixed

#### Core

- Improved validation of incoming tuples. Now tuples coming over the network can't contain malformed decimals, uuids, or datetime values (gh-6857).
- Fixed a bug in the net.box connector because of which a client could fail to close its connection when receiving a shutdown request from the server. This could lead to the server hanging on exit (gh-7225).
- Fixed a crash and possible undefined behaviour when using `scalar` and `number` indexes over fields containing both decimals and double `Inf` or `NaN`.

  For vinyl spaces, the above conditions could lead to wrong ordering of indexed values. To fix the issue, recreate the indexes on such spaces following this guide (gh-6377).
- Fixed a bug because of which a net.box connection was not properly terminated when the process had a child (for example, started with `popen`) sharing the connection socket fd. The bug could lead to a server hanging on exit while executing the graceful shutdown protocol (gh-7256).
- Removed an assertion on `fiber_wakeup()` calls with dead fibers in debug builds. Such behavior was inconsistent with release builds, in which the same calls were allowed (gh-5843).
- Fixed the `exclude_null` index option not working for multikey and JSON indexes (gh-5861).
- Fixed the performance degradation of fiber backtrace collection after the backtrace rework (gh-7207).
- Fixed a hang when a synchronous request was issued from a net.box `on_connect` or `on_schema_reload` trigger. Now an error is raised instead (gh-5358).
- Fixed a crash that could happen on x86 systems without the `RDTSCP` instruction (gh-5869).
- Fixed a bug that allowed to access indexed fields of nested tuples with `[*]` in Lua (gh-5226).

- Fixed the behavior of `space_object:fselect()` on binary data (gh-7040).
- Fixed Tarantool not being able to recover from old snapshots when `box.cfg.work_dir` and `box.cfg.memtx_dir` were both set (gh-7232).
- Fixed Tarantool terminations on error messages with invalid UTF-8 sequences (gh-6781 and gh-6934).
- Fixed a bug when the `Transaction is active at return from function` error was overwriting expression evaluation errors in case the expression begins a transaction (gh-7288).
- Added type checking for options in net.box's remote queries and connect method. Now graceful errors are thrown in case of incorrect options (gh-6063, gh-6530).
- Fixed `space_object:format()` and `space_object.foreign_key` returning incorrect numbers of foreign key fields (gh-7350).
- Fixed the foreign key check on `space_object:truncate()` calls (gh-7309).
- Fixed a crash when `box.stat.net.thread[i]` is called with invalid `i` values (gh-7196).
- Fixed a low-probability stack overflow bug in the qsort implementation.

### Memtx

- Fixed the ability to perform read-only operations in conflicting transactions in memtx, which led to spurious results (gh-7238).
- Fixed false assertion on repeatable `replace` with the memtx transaction manager enabled (gh-7214).
- Fixed false transaction conflict on repeatable `insert`/`upsert` with the memtx transaction manager enabled (gh-7217).
- Fixed dirty reads in the `GT` iterator of HASH indexes (gh-7477).
- Fixed phantom reads in reverse iterators (gh-7409).
- Fixed `select` with `LE` iterator in **memtx TREE** index returning deleted tuple (gh-7432).
- Fixed incorrect handling of corner cases gap tracking in transaction manager (gh-7375).
- Fixed a bug in the memtx hash index implementation that could lead to uncommitted data written to a snapshot file (gh-7539).

### Vinyl

- Fixed a bug in the vinyl upsert squashing optimization that could lead to a segmentation fault error (gh-5080).
- Fixed a bug in the vinyl garbage collector. It could skip stale tuples stored in a secondary index if upsert operations were used on the space before the index was created (gh-3638).
- Fixed a bug in the vinyl read iterator that could result in a significant performance degradation of range select requests in the presence of an intensive write workload (gh-5700).
- Explicitly disabled the hot standby mode for vinyl. Now an attempt to enable the hot standby mode in case the master instance has vinyl spaces results in an error. Before this change, the behavior was undefined (gh-6565).

### Replication

- Added the logging of the error reason on a replica in case when the master didn't send a greeting message (gh-7204).
- Fixed replication being stuck occasionally for no obvious reasons.
- Fixed a possible split-brain when the old synchro queue owner might finalize the transactions in the presence of the new owner (gh-5295).
- Improved the detection of possible split-brain situations, for example, when multiple leaders were working independently due to manually lowered quorum. Once a node discovers that it received some foreign data, it immediately stops replication from such a node with an `ER_SPLIT_BRAIN` error (gh-5295).
- Fixed a false positive split-brain error after `box.ctl.demote()` (gh-7286).
- Fixed a bug when followers with `box.cfg.election_mode` turned on did not notice the leader hang due to a long request, such as a `select{}` from a large space or a `pairs` iteration without yields between loop cycles (gh-7512).

### LuaJIT

Backported patches from vanilla LuaJIT trunk (gh-6548 and gh-7230). In the scope of this activity, the following issues have been resolved:

- Fixed emitting for fuse load of constant in GC64 mode (gh-4095, gh-4199, gh-4614).
- Now initialization of zero-filled struct is compiled (gh-4630, gh-5885).
- Actually implemented `maxirconst` option for tuning JIT compiler.
- Fixed JIT stack of Lua slots overflow during recording for metamethod calls.
- Fixed bytecode dump unpatching for JLOOP in up-recursion compiled functions.
- Fixed FOLD rule for strength reduction of widening in cdata indexing.
- Fixed `string.char()` recording without arguments.
- Fixed `print()` behaviour with the reloaded default metatable for numbers.
- `tonumber("-0")` now saves the sign of number for conversion.
- `tonumber()` now gives predictable results for negative non-base-10 numbers.
- Fixed write barrier for `debug.setupvalue()` and `lua_setupvalue()`.
- Fixed conflict between 64 bit lightuserdata and ITERN key for ARM64.
- Fixed emitting assembly for HREFK on ARM64.
- Fixed pass-by-value struct in FFI calls on ARM64.
- `jit.p` now flushes and closes output file after run, not at program exit.
- Fixed `jit.p` profiler interaction with GC finalizers.
- Fixed the case for partial recording of vararg function body with the fixed number of result values in with `LJ_GC64` (i.e. `LJ_FR2` enabled) (gh-7172).
- Added `/proc/self/exe` symlink resolution to the symtab module to obtain the `.symtab` section for the Tarantool executable.
- Introduced stack sandwich support to sysprof's parser (gh-7244).

- Disabled proto and trace information dumpers in sysprof's default mode. Attempts to use them lead to a segmentation fault due to an uninitialized buffer (gh-7264).
- Fixed handling of errors during trace snapshot restore.

#### Lua

- The `fiber_obj:info()` now correctly handles its options (gh-7210).
- Fixed a bug when Ctrl+C doesn't discard the multiline input (gh-7109).

#### SQL

- Fixed the creation of ephemeral space format in `ORDER BY` (gh-7043).
- The result type of arithmetic operations between two unsigned values is now `INTEGER` (gh-7295).
- Fixed a bug with the `ANY` type in the ephemeral space format in `ORDER BY` (gh-7043).
- Truncation of a space no longer corrupts prepared statements (gh-7358).

#### Datetime

- Fixed a bug when `date:set{hour=nil,min=XXX}` did not retain the original `hour` value (gh-7298).
- Introduced the validation of incoming data at the moment messagepack is converted to datetime (gh-6723).

#### HTTP client

- Enabled the automatic detection of system CA certificates in the runtime (gh-7372). It was disabled in 2.10.0, which led to the inability to use HTTPS without the `verify_peer = false` option.

#### Build

- Fixed a build failure with gcc if libpbf is installed (gh-7292).
- Fixed the static build on Mac OS 11 and newer (gh-7459).

### Tarantool 2.10.0

Released on 2022-05-22.

- Release: v. 2.10.0.
- Tag: `2.10.0`.

### Overview

2.10.0 is the first *stable* version of the 2.10 release series. It introduces 107 improvements and resolves 131 bugs since version *2.8.1*.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

Notable changes are:

- HTTP client now supports HTTP/2.
- Support of the new DATETIME type.
- Improved type consistency in SQL.
- Added transaction isolation levels.
- Implemented fencing and pre-voting in RAFT.
- Introduced foreign keys and constraints.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please *upgrade* using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

Some changes are labeled as **[Breaking change]**. It means that the old behavior was considered error-prone and therefore changed to protect users from unintended mistakes. However, there is a small probability that someone can rely on the old behavior, and this label is to bring attention to the things that have been changed.

The changes that break backward compatibility are listed below:

- `fiber.wakeup()` in Lua and `fiber_wakeup()` in C became NOP on the currently running fiber.
- The `timeout()` method of `net.box` connection was dropped.
- The net.box console support was dropped.
- The return value type for all time64 functions was changed from `uint64_t` to `int64_t`.

### Functionality added or changed

#### Core

- The `UUID` field type is now part of the `SCALAR` field type (gh-6042).
- The `UUID` field type is now available in SQL. A new `UUID` can be generated using the new SQL built-in `UUID()` function (gh-5886).
- **[Breaking change]** The `timeout()` method of `net.box` connection, marked deprecated more than four years ago (in `1.7.4`), has been dropped. It negatively affected the performance of hot `net.box` methods, such as `call()` and `select()` if they were called without specifying a timeout (gh-6242).
- Improved `net.box` performance by up to 70% by rewriting hot code paths in C (gh-6241).
- Introduced compact tuples that allow saving 4 bytes per tuple in case of small user data (gh-5385).

- Now streams and interactive transactions over streams are implemented in iproto. Every stream is associated with its ID, which is unique within one connection. All requests with the same non-zero stream ID belong to the same stream. All requests in the stream are processed synchronously. The next request will not start executing until the previous one is completed. If a request's stream ID is `0`, it does not belong to a stream and is processed in the old way.

  In `net.box`, a stream is an object above the connection that has the same methods but allows executing requests sequentially. The ID is generated on the client-side automatically. If a user writes his own connector and wants to use streams, they must transmit the `stream_id` over the iproto protocol.

  The primary purpose of streams is transactions via iproto. As each stream can start a transaction, several transactions can be multiplexed over one connection. There are multiple ways to begin, commit, and rollback a transaction. One can do that using the appropriate stream methods, `call`, `eval`, or `execute` with the SQL transaction syntax. Users can mix these methods. For example, one might start a transaction using `stream:begin()`, and commit it with `stream:call('box.commit')` or `stream:execute('COMMIT')`.

  If any request fails during the transaction, it will not affect the other requests in the transaction. If a disconnect occurs while there is an active transaction in the stream, that transaction will be rolled back if it hasn't been committed before the connection failure (gh-5860).

- Added the new `memtx_allocator` option to `box.cfg{}`. It allows selecting the appropriate allocator for memtx tuples if necessary. The possible values are `system` for malloc allocator and `small` for the default small allocator.

  Implemented the system allocator based on malloc. The slab allocator, which is used for tuple allocation, has a particular disadvantage – it is prone to unresolvable fragmentation on specific workloads (size migration). In this case, the user should be able to choose another allocator. The system allocator is based on the malloc function but restricted by the same quota as the slab allocator. The system allocator does not alloc all the memory at the start. Instead, it allocates memory as needed, checking that the quota is not exceeded (gh-5419).

- Added `box.stat.net.thread()` for reporting per thread net statistics (gh-6293).

- Added the new `STREAMS` metric to `box.stat.net`. It contains statistics for iproto streams. The `STREAMS` contains the same counters as the `CONNECTIONS` metric in `box.stat.net`: `current`, `RPS`, and `total` (gh-6293).

- Extended the network protocol (IPROTO) with a new request type (`IPROTO_ID`). It is supposed to be used for exchanging sets of supported features between the server and client (gh-6253).

- Added `required_protocol_version` and `required_protocol_features` to `net.box` connection options. The new options allow specifying the IPROTO protocol version and features that must be supported by the server for the connection to pass (gh-6253).

- **[Breaking change]** Added the `msgpack.cfg.encode_error_as_ext` configuration option to enable/disable encoding errors as `MP_ERROR` MsgPack extension. The option is enabled by default (gh-6433).

- **[Breaking change]** Removed `box.session.setting.error_marshaling_enabled`. Error marshalling is now enabled automatically if the connector supports it (gh-6428).

- Added the new `REQUESTS_IN_PROGRESS` and `REQUESTS_IN_STREAM_QUEUE` metrics to `box.stat.net` that contains detailed statistics for iproto requests. These metrics contain the same counters as other metrics in `box.stat.net`: `current`, `RPS`, and `total` (gh-6293).

- Implemented a timeout for `fiber:join` in Lua (gh-6203).

- Updated libev to version 4.33 (gh-4909).

- Added the new `box.txn_id()` function. It returns the ID of the current transaction if called within a transaction, nil otherwise.
- Previously, if a yield occurs for a transaction that does not support it, all its statements are rolled back but still its new statements are processed (they will roll back with each yield). Also, the transaction will be rolled back when a commit is attempted. Now we stop processing any new statements right after the first yield if a transaction does not support it.
- Implemented a timeout for transactions after which they are rolled back (gh-6177).
- Implemented the new C API `box_txn_set_timeout` function to set a timeout for transactions.
- Implemented a timeout for iproto transactions after which they are rolled back (gh-6177).
- Implemented the new `IPROTO_TIMEOUT 0x56` key, which is used to set a timeout for transactions over iproto streams. It is stored in the body of `IPROTO_BEGIN` request.
- Introduced `box.broadcast` and `box.watch` functions to signal/watch user-defined state changes (gh-6257).
- Added watchers support to the network protocol (gh-6257).
- Added watchers support to the `net.box` connector (gh-6257).
- Now error objects with the code `box.error.READONLY` now have the additional fields explaining why the error happened.

  Also, there is a new field `box.info.ro_reason`. It is `nil` on a writable instance, but reports a reason when `box.info.ro` is `true` (gh-5568).
- Implemented the ability to open several listening sockets. In addition to the ability to pass uri as a number or string, added the ability to pass uri as a table of numbers or strings (gh-3554).
- **[Breaking change]** `net.box` console support, which was marked deprecated in 1.10, has been dropped. Use `require('console').connect()` instead.
- Added the `takes_raw_args` Lua function option for wrapping arguments in `msgpack.object` to skip decoding (gh-3349).
- Implemented the graceful shutdown protocol for IPROTO connections (gh-5924).
- Added `fetch_schema` flag to `netbox.connect` to control schema fetching from remote instance (gh-4789).
- Added linking type (dynamic or static) to Tarantool build info.
- Changed log level of some information messages from critical to info (gh-4675).
- Added predefined system events: `box.status`, `box.id`, `box.election`, and `box.schema` (gh-6260).
- Introduced transaction isolation levels in Lua and IPROTO (gh-6930).
- Added support for backtrace feature on AARCH64 architecture (gh-6060).
- Implemented collection of parent backtrace for the newly created fibers. To enable the feature, call `fiber.parent_backtrace_enable`. To disable it, call `fiber.parent_backtrace_disable`: disabled by default (gh-4302).

### Memtx

- Introduced memtx MVCC memory monitoring (gh-6150).

### Vinyl

- Disabled the deferred `DELETE` optimization in Vinyl to avoid possible performance degradation of secondary index reads. Now, to enable the optimization, one has to set the `defer_deletes` flag in space options (gh-4501).

### Replication

- Introduced `box.info.replication[n].downstream.lag` field to monitor the state of replication. This member represents a lag between the main node writing a certain transaction to its own WAL and the moment it receives an ack for this transaction from a replica (gh-5447).
- Introduced `on_election` triggers. The triggers may be registered via `box.ctl.on_election()` interface and are run asynchronously each time `box.info.election` changes (gh-5819).
- It is now possible to decode incoming replication data in a separate thread. Added the `replication_threads` configuration option that controls how many threads may be spawned to do the task (default is 1) (gh-6329).

### Raft

- Added the `term` field to `box.info.synchro.queue`. It contains a term of the last `PROMOTE`. It is usually equal to `box.info.election.term` but may be less than the election term when the new round of elections started, but no one promoted yet.
- Servers with elections enabled won't start new elections as long as at least one of their peers sees the current leader. They also won't start the elections when they don't have a quorum of connected peers. This should reduce cases when a server that has lost connectivity to the leader disrupts the whole cluster by starting new elections (gh-6654).
- Added the `leader_idle` field to `box.info.election` table. The value shows time in seconds since the last communication with a known leader (gh-6654).

### LuaJIT

- Introduced support for `LJ_DUALNUM` mode in `luajit-gdb.py` (gh-6224).
- Introduced preliminary support of GNU/Linux ARM64 and macOS M1. In the scope of this activity, the following issues have been resolved:
  - Introduced support for a full 64-bit range of lightuserdata values (gh-2712).
  - Fixed memory remapping issue when the page leaves 47-bit segments.
  - Fixed M1 architecture detection (gh-6065).
  - Fixed variadic arguments handling in FFI on M1 (gh-6066).
  - Fixed `table.move` misbehavior when table reallocation occurs (gh-6084).

  - Fixed Lua stack inconsistency when xpcall is called with an invalid second argument on ARM64 (gh-6093).
  - Fixed `BC_USETS` bytecode semantics for closed upvalues and gray strings.
  - Fixed side exit jump target patching considering the range values of the particular instruction (gh-6098).
  - Fixed current Lua coroutine restoring on an exceptional path on ARM64 (gh-6189).

- Now memory profiler records allocations from traces grouping them by the trace number (gh-5814). The memory profiler parser can display the new type of allocation sources in the following format:

```
| TRACE [<trace-no>] <trace-addr> started at @<sym-chunk>:<sym-line>
```

- Now the memory profiler reports allocations made by the JIT engine while compiling the trace as INTERNAL (gh-5679).
- Now the memory profiler emits events of the new type when a function or a trace is created. As a result, the memory profiler parser can enrich its symbol table with the new functions and traces (gh-5815).

  Furthermore, there are symbol generations introduced within the internal parser structure to handle possible collisions of function addresses and trace numbers.
- Now the memory profiler dumps symbol table for C functions. As a result, memory profiler parser can enrich its symbol table with C symbols (gh-5813). Furthermore, now memory profiler dumps special events for symbol table when it encounters a new C symbol, that has not been dumped yet.
- Introduced the LuaJIT platform profiler (gh-781) and the profile parser. This profiler is able to capture both host and VM stacks, so it can show the whole picture. Both C and Lua API's are available for the profiler. Profiler comes with the default parser, which produces output in a `flamegraph.pl`-suitable format. The following profiling modes are available:
  - Default: only virtual machine state counters.
  - Leaf: shows the last frame on the stack.
  - Callchain: performs a complete stack dump.

### Lua

- Introduced the new method `table.equals`. It compares two tables by value and respects the `__eq` metamethod.
- Added support of console autocompletion for `net.box` objects `stream` and `future` (gh-6305).
- Added the `box.runtime.info().tuple` metric to track the amount of memory occupied by tuples allocated on runtime arena (gh-5872).

  It does not count tuples that arrive from memtx or vinyl but counts tuples created on-the-fly: say, using `box.tuple.new(<...>)`.

### Datetime

- Added a new built-in module `datetime.lua` that allows operating timestamps and intervals values (gh-5941).
- Added the method to allow converting string literals in extended iso-8601 or rfc3339 formats (gh-6731).
- Extended the range of supported years in all parsers to cover fully -5879610-06-22..5879611-07-11 (gh-6731).
- Datetime interval support has been reimplemented in C to make possible future Olson/tzdata and SQL extensions (gh-6923).

  Now all components of the interval values are kept and operated separately (years, months, weeks, days, hours, seconds, and nanoseconds). This allows applying date/time arithmetic correctly when we add/subtract intervals to datetime values.
- Extended datetime literal parser with the ability to handle known timezone abbreviations ('MSK', 'CET', etc.) which are deterministically translated to their offset (gh-5941, gh-6751).

  Timezone abbreviations can be used in addition to the timezone offset in the datetime literals. For example, these literals produce equivalent datetime values:

```
local date = require('datetime')
local d1 = date.parse('2000-01-01T02:00:00+0300')
local d2 = date.parse('2000-01-01T02:00:00 MSK')
local d3 = date.parse('2000-01-01T02:00:00 MSK', {format = '%FT%T %Z'})
```

  Parser fails if one uses ambiguous names (for example, 'AT') which could not be directly translated into timezone offsets.
- Enabled support for timezone names in the constructor and `date:set{}` modifier via `tz` attribute. Currently, only timezone name abbreviations are supported (gh-7076).

  Timezone abbreviations can be used in addition to the timezone offset. They can be used during constructing or modifying a date object, or while parsing datetime literals. Numeric time offsets and named abbreviations produce equivalent datetime values:

```
local date = require('datetime')
local d2 = date.parse('2000-01-01T02:00:00 MSK')

local d1 = date.new{year = 1980, tz = 'MSK'}
d2 = date.new{year = 1980, tzoffset = 180}
d2:set{tz = 'MSK'}
```

  Note that the timezone name parser fails if one uses ambiguous names, which could not be translated into timezone offsets directly (for example, 'AT').

### Digest

- Introduced new hash types in digest module – `xxhash32` and `xxhash64` (gh-2003).

### Fiber

- Introduced `fiber_object:info()` to get `info` from fiber. Works as `require('fiber').info()` but only for one fiber.
- Introduced `fiber_object:csw()` to get `csw` from fiber (gh-5799).
- Changed `fiber.info()` to hide backtraces of idle fibers (gh-4235).
- Improved fiber `fiber.self()`, `fiber.id()` and `fiber.find()` performance by 2-3 times.

### Log

- Implemented support of symbolic log levels representation in `log` module (gh-5882). Now it is possible to specify levels the same way as in `box.cfg{}` call.

  For example, instead of

  ```
  require('log').cfg{level = 6}
  ```

  one can use

  ```
  require('log').cfg{level = 'verbose'}
  ```

### Msgpack

- Added the `msgpack.object` container for marshaling arbitrary MsgPack data (gh-1629, gh-3349, gh-3909, gh-4861, gh-5316).

### Netbox

- Added the `return_raw` net.box option for returning `msgpack.object` instead of decoding the response (gh-4861).

### Schema

- `is_multikey` option may now be passed to `box.schema.func.create` directly, without `opts` sub-table.

### SQL

- Descriptions of type mismatch error and inconsistent type error became more informative (gh-6176).
- Removed explicit cast from `BOOLEAN` to numeric types and vice versa (gh-4770).
- Removed explicit cast from `VARBINARY` to numeric types and vice versa (gh-4772, gh-5852).
- Fixed a bug due to which a string that is not `NULL`-terminated could not be cast to `BOOLEAN`, even if the conversion should be successful according to the rules.
- Now a numeric value can be cast to another numeric type only if the cast is precise. In addition, a `UUID` value cannot be implicitly cast to `STRING`/`VARBINARY`. Also, a `STRING`/`VARBINARY` value cannot be implicitly cast to a `UUID` (gh-4470).
- Now any number can be compared to any other number, and values of any scalar type can be compared to any other value of the same type. A value of a non-numeric scalar type cannot be compared with a value of any other scalar type (gh-4230).
- SQL built-in functions were removed from the `_func` system space (gh-6106).
- Functions are now looked up first in SQL built-in functions and then in user-defined functions.
- Fixed incorrect error message in case of misuse of the function used to set the default value.
- The `typeof()` function with `NULL` as an argument now returns `NULL` (gh-5956).
- The `SCALAR` and `NUMBER` types have been reworked in SQL. Now `SCALAR` values cannot be implicitly cast to any other scalar type, and `NUMBER` values cannot be implicitly cast to any other numeric type. This means that arithmetic and bitwise operations and concatenation are no longer allowed for `SCALAR` and `NUMBER` values. In addition, any `SCALAR` value can now be compared with values of any other scalar type using the `SCALAR` rules (gh-6221).
- The `DECIMAL` field type is now available in SQL. Decimal can be implicitly cast to and from `INTEGER` and `DOUBLE`, it can participate in arithmetic operations and comparison between `DECIMAL`, and all other numeric types are defined (gh-4415).
- The argument types of SQL built-in functions are now checked in most cases during parsing. In addition, the number of arguments is now always checked during parsing (gh-6105).
- `DECIMAL` values can now be bound in SQL (gh-4717).
- A value consisting of digits and a decimal point is now parsed as `DECIMAL` (gh-6456).
- The `ANY` field type is now available in SQL (gh-3174).
- Built-in SQL functions now work correctly with `DECIMAL` values (gh-6355).
- The default type is now defined in case the argument type of an SQL built-in function cannot be determined during parsing (gh-4415).
- The `ARRAY` field type is now available in SQL. The syntax has also been implemented to allow the creation of `ARRAY` values (gh-4762).
- User-defined aggregate functions are now available in SQL (gh-2579).
- Introduced SQL built-in functions `NOW()` and `DATE_PART()` (gh-6773).
- The left operand is now checked before the right operand in an arithmetic operation. (gh-6773).
- The `INTERVAL` field type is introduced in SQL (gh-6773).
- Bitwise operations can now only accept `UNSIGNED` and positive `INTEGER` values (gh-5364).
- The `MAP` field type is now available in SQL. Also, the syntax has been implemented to allow the creation of `MAP` values (gh-4763).

- Introduced `[]` operator for `MAP` and `ARRAY` values (gh-6251).

### Box

- Public role now has read, write access on `_session_settings` space (gh-6310).
- The `INTERVAL` field type is introduced to `BOX` (gh-6773).
- The behavior of empty or nil `select` calls on user spaces was changed. A critical log entry containing the current stack traceback is created upon such function calls. The user can explicitly request a full scan though by passing `fullscan=true` to `select` 's `options` table argument, in which case a log entry will not be created (gh-6539).
- Improved checking for dangerous `select` calls. The calls with `offset + limit <= 1000` are now considered safe, which means a warning is not issued. The 'ALL', 'GE', 'GT', 'LE', 'LT' iterators are now considered dangerous by default even with the key present (gh-7129).

### Datetime

- Allowed using human-readable timezone names (for example, 'Europe/Moscow') in datetime strings. Use IANA `tzdata` (Olson DB) for timezone-related operations, such as DST-based timezone offset calculations (gh-6751).
- The `isdst` field in the datetime object is now calculated correctly, according to the IANA `tzdata` (aka Olson DB) rules for the given date/time moment (gh-6751).
- The `datetime` module exports the bidirectional `TZ` array, which can be used to translate the timezone index (`tzindex`) into timezone names, and vice versa (gh-6751).

### Fiber

- Previously csw (Context SWitch) of a new fiber could be more than 0, now it is always 0 (gh-5799).

### Luarocks

- Set `FORCE_CONFIG=false` for luarocks config to allow loading project-side `.rocks/config-5.1.lua`.

### Xlog

- Reduced snapshot verbosity (gh-6620).

### Build

- Support fedora-34 build (gh-6074).
- Stopped support fedora-28 and fedora-29.
- Stopped support of Ubuntu Trusty (14.04) (gh-6502).
- Bumped Debian package compatibility level to 10 (gh-5429).
- Bumped minimal required debhelper to version 10 (except for Ubuntu Xenial).

- Removed Windows binaries from Debian source packages (gh-6390).
- Bumped Debian control Standards-Version to 4.5.1 (gh-6390).
- Added bundling of libnghttp2 for bundled libcurl to support HTTP/2 for http client. The CMake version requirement is updated from 3.2 to 3.3.
- Support fedora-35 build (gh-6692).
- Added bundling of GNU libunwind to support backtrace feature on AARCH64 architecture and distributives that don't provide libunwind package.
- Re-enabled backtrace feature for all RHEL distributions by default, except for AARCH64 architecture and ancient GCC versions, which lack compiler features required for backtrace (gh-4611).
- Updated `libicu` version to 71.1 for static build.
- Bumped OpenSSL from 1.1.1f to 1.1.1n for static build (gh-6947).
- Updated libcurl to version 7.83.0 (gh-6029).
- Support Fedora-36 build.
- Support Ubuntu Jammy (22.04) build.

### Bugs fixed

#### Core

- **[Breaking change]** `fiber.wakeup()` in Lua and `fiber_wakeup()` in C became NOP on the currently running fiber.

  Previously they allowed ignoring the next yield or sleep, which resulted in unexpected erroneous wake-ups. Calling these functions right before `fiber.create()` in Lua or `fiber_start()` in C could lead to a crash (in debug build) or undefined behaviour (in release build) (gh-6043).

  There was a single use case for that—reschedule in the same event loop iteration which is not the same as `fiber.sleep(0)` in Lua and `fiber_sleep(0)` in C. It could be done in the following way:

  in C:

  ```
  fiber_wakeup(fiber_self());
  fiber_yield();
  ```

  in Lua:

  ```
  fiber.self():wakeup()
  fiber.yield()
  ```

  To get the same effect in C, one can use `fiber_reschedule()`. In Lua, it is now impossible to reschedule the current fiber directly in the same event loop iteration. One can reschedule self through a second fiber, but it is strongly discouraged:

  ```
  local self = fiber.self()
  fiber.new(function() self:wakeup() end)
  fiber.sleep(0)
  ```

- Fixed memory leak on each `box.on_commit()` and `box.on_rollback()` (gh-6025).

- Fixed the lack of testing for non-joinable fibers in `fiber_join()` call. This could lead to unpredictable results. Note the issue affects C level only, in Lua interface `fiber:join()` the protection is turned on already.
- Now Tarantool yields when scanning `.xlog` files for the latest applied vclock and when finding the right place in `.xlog`s to start recovering. This means that the instance is responsive right after `box.cfg` call even when an empty `.xlog` was not created on the previous exit. Also, this prevents the relay from timing out when a freshly subscribed replica needs rows from the end of a relatively long (hundreds of MBs) `.xlog` (gh-5979).
- The counter in `x.yM rows processed` log messages does not reset on each new recovered `xlog` anymore.
- Fixed wrong type specification when printing fiber state change which led to negative fiber's ID logging (gh-5846).

  For example,

  ```
  main/-244760339/cartridge.failover.task I> Instance state changed
  ```

  instead of proper

  ```
  main/4050206957/cartridge.failover.task I> Instance state changed
  ```

- Fiber IDs were switched to monotonically increasing unsigned 8-byte integers so that there would not be IDs wrapping anymore. This allows detecting fiber's precedence by their IDs if needed (gh-5846).
- Fixed a crash in JSON update on tuple/space when it had more than one operation, they accessed fields in reversed order, and these fields did not exist. Example: `box.tuple.new({1}):update({{'=', 4, 4}, {'=', 3, 3}})` (gh-6069).
- Fixed invalid results produced by the `json` module's `encode` function when it was used from Lua's garbage collector. For instance, in functions used as `ffi.gc()` (gh-6050).
- Added check for user input of the number of iproto threads—value must be $> 0$ and less than or equal to 1000 (gh-6005).
- Fixed error related to the fact that if a user changed the listen address, all iproto threads closed the same socket multiple times.
- Fixed error related to Tarantool not deleting the unix socket path when the work is finished.
- Fixed a crash in MVCC during simultaneous update of a key in different transactions (gh-6131).
- Fixed a bug when memtx MVCC crashed during reading uncommitted DDL (gh-5515).
- Fixed a bug when memtx MVCC crashed if an index was created in the transaction (gh-6137).
- Fixed segmentation fault with MVCC when an entire space was updated concurrently (gh-5892).
- Fixed a bug with failed assertion after stress update of the same key (gh-6193).
- Fixed a crash that happened when a user called `box.snapshot` during an incomplete transaction (gh-6229).
- Fixed console client connection breakage if request times out (gh-6249).
- Added missing broadcast to `net.box.future:discard()`. Now fibers waiting for a request result are woken up when the request is discarded (gh-6250).
- `box.info.uuid`, `box.info.cluster.uuid`, and `tostring(decimal)` with any decimal number in Lua sometimes could return garbage if `__gc` handlers were used in the user's code (gh-6259).
- Fixed the error message that happened in a very specific case during MVCC operation (gh-6247).

- Fixed a repeatable read violation after delete (gh-6206).
- Fixed a bug when hash `select{}` was not tracked by MVCC engine (gh-6040).
- Fixed a crash in MVCC after the drop of a space with several indexes (gh-6274).
- Fixed a bug when GC at some state could leave tuples in secondary indexes (gh-6234).
- Disallowed yields after DDL operations in MVCC mode. It fixes a crash which takes place in case several transactions refer to system spaces (gh-5998).
- Fixed a bug in MVCC connected which happened on a rollback after DDL operation (gh-5998).
- Fixed a bug when rollback resulted in unserializable behaviour (gh-6325).
- At the moment, when a `net.box` connection is closed, all requests that have not been sent will be discarded. This patch fixes this behavior: all requests queued for sending before the connection is closed are guaranteed to be sent (gh-6338).
- Fixed a crash during replace of malformed tuple into `_schema` system space (gh-6332).
- Fixed dropping incoming messages when the connection is closed or `SHUT_RDWR` received and `net_msg_max` or readahead limit is reached (gh-6292).
- Fixed memory leak in case of replace during background alter of the primary index (gh-6290).
- Fixed a bug when rolled back changes appear in the built-in-background index (gh-5958).
- Fixed a crash while encoding an error object in the MsgPack format (gh-6431).
- Fixed a bug when an index was inconsistent after background build in case the primary index was hash (gh-5977).
- Now inserting a tuple with the wrong `id`` field into the `_priv` space returns the correct error (gh-6295).
- Fixed dirty read in MVCC after space alter (gh-6263, gh-6318).
- Fixed a crash in case the fiber changing `box.cfg.listen` is woken up (gh-6480).
- Fixed `box.cfg.listen` not reverted to the old address in case the new one is invalid (gh-6092).
- Fixed a crash caused by a race between `box.session.push()` and closing connection (gh-6520).
- Fixed a bug because of which the garbage collector could remove an `xlog` file that was still in use (gh-6554).
- Fixed crash during granting privileges from guest (gh-5389).
- Fixed an error in listening when the user passed uri in numerical form after listening unix socket (gh-6535).
- Fixed a crash that could happen in case a tuple is deleted from a functional index while there is an iterator pointing to it (gh-6786).
- Fixed memory leak in interactive console (gh-6817).
- Fixed an assertion fail when passing a tuple without primary key fields to `before_replace` trigger. Now tuple format is checked before execution of `before_replace` triggers and after each one (gh-6780).
- Banned DDL operations in space `on_replace` triggers, since they could lead to a crash (gh-6920).
- Implemented constraints and foreign keys. Now users can create function constraints and foreign key relations (gh-6436).
- Fixed a bug due to which all fibers created with `fiber_attr_setstacksize()` leaked until the thread exit. Their stacks also leaked except when `fiber_set_joinable(..., true)` was used.
- Fixed a crash in MVCC related to a secondary index conflict (gh-6452).

- Fixed a bug which resulted in wrong space count (gh-6421).
- `SELECT` in RO transaction now reads confirmed data, like a standalone (autocommit) `SELECT` does (gh-6452).
- Fixed a crash when Tarantool was launched with multiple `-e` or `-l` options without a space between the option and the value (gh-5747).
- Fixed effective session and user not propagated to `box.on_commit` and `box.on_rollback` trigger callbacks (gh-7005).
- Fixed usage of `box.session.peer()` in `box.session.on_disconnect()` trigger. Now it's safe to assume that `box.session.peer()` returns the address of the disconnected peer, not nil, as it used to (gh-7014).
- Fixed creation of a space with a foreign key pointing to the same space (gh-6961).
- Fixed a bug when MVCC failed to track nothing-found range `select` (gh-7025).
- Allowed complex foreign keys with NULL fields (gh-7046).
- Added decoding of election messages: `RAFT` and `PROMOTE` to `xlog` Lua module (gh-6088). Otherwise `tarantoolctl` shows plain number in `type`

```
HEADER:
  lsn: 1
  replica_id: 4
  type: 31
  timestamp: 1621541912.4592
```

  instead of symbolic representation

```
HEADER:
  lsn: 1
  replica_id: 4
  type: PROMOTE
  timestamp: 1621541912.4592
```

- **[Breaking change]** Return value signedness of 64-bit time functions in `clock` and `fiber` was changed from `uint64_t` to `int64_t` both in Lua and C (gh-5989).
- Fixed reversed iterators gap tracking. Instead of tracking gaps for the successors of keys, gaps for tuples shifted by one to the left of the successor were tracked (gh-7113).

### Memtx

- Now memtx raises an error if the "clear" dictionary is passed to `s:select()` (gh-6167).
- Fixed MVCC transaction manager story garbage collection breaking memtx TREE index iterator (gh-6344).

### Vinyl

- Fixed possible keys divergence during secondary index build, which might lead to missing tuples (gh-6045).
- Fixed the race between Vinyl garbage collection and compaction that resulted in a broken vylog and recovery failure (gh-5436).
- Immediate removal of compacted run files created after the last checkpoint optimization now works for replica's initial JOIN stage (gh-6568).
- Fixed crash during recovery of a secondary index in case the primary index contains incompatible phantom tuples (gh-6778).

### Replication

- Fixed the use after free in the relay thread when using elections (gh-6031).
- Fixed a possible crash when a synchronous transaction was followed by an asynchronous transaction right when its confirmation was being written (gh-6057).
- Fixed an error where a replica, while attempting to subscribe to a foreign cluster with a different replicaset UUID, did not notice it is impossible and instead became stuck in an infinite retry loop printing a `TOO_EARLY_SUBSCRIBE` error (gh-6094).
- Fixed an error where a replica, while attempting to join a cluster with exclusively read-only replicas available, just booted its own replicaset, instead of failing or retrying. Now it fails with an error about the other nodes being read-only so they can't register the new replica (gh-5613).
- Fixed error reporting associated with transactions received from remote instances via replication. Any error raised while such a transaction was being applied was always reported as `Failed to write to disk` regardless of what really happened. Now the correct error is shown. For example, `Out of memory`, or `Transaction has been aborted by conflict`, and so on (gh-6027).
- Fixed replication stopping occasionally with `ER_INVALID_MSGPACK` when replica is under high load (gh-4040).
- Fixed a cluster that sometimes could not bootstrap if it contained nodes with `election_mode manual` or `voter` (gh-6018).
- Fixed a possible crash when `box.ctl.promote()` was called in a cluster with >= 3 instances, happened in debug build. In release build, it could lead to undefined behavior. It was likely to happen if a new node was added shortly before the promotion (gh-5430).
- Fixed a rare error appearing when MVCC (`box.cfg.memtx_use_mvcc_engine`) was enabled and more than one replica was joined to a cluster. The join could fail with the error `"ER_TUPLE_FOUND: Duplicate key exists in unique index 'primary' in space '_cluster'"`. The same could happen at the bootstrap of a cluster having >= 3 nodes (gh-5601).
- Fixed replica reconnecting to a living master on any `box.cfg{replication=...}` change. Such reconnects could lead to replica failing to restore connection for `replication_timeout` seconds (gh-4669).
- Fixed potential obsolete data write in synchronous replication due to race in accessing terms while disk write operation is in progress and not yet completed.
- Fixed replicas failing to bootstrap when the master has just restarted (gh-6966).
- Fixed a bug when replication was broken on the master side with `ER_INVALID_MSGPACK` (gh-7089).

### Raft

- Fixed `box.ctl.promote()` entering an infinite election loop when a node does not have enough peers to win the elections (gh-6654).
- Servers with elections enabled will resign the leadership and become read-only when the number of connected replicas becomes less than a quorum. This should prevent split-brain in some situations (gh-6661).
- Fixed a rare crash with the leader election enabled (any mode except `off`), which could happen if a leader resigned from its role at the same time as some other node was writing something related to the elections to WAL. The crash was in debug build. In the release build, it would lead to undefined behavior (gh-6129).
- Fixed an error when a new replica in a Raft cluster could try to join from a follower instead of a leader and failed with an error `ER_READONLY` (gh-6127).
- Reconfiguration of `box.cfg.election_timeout` could lead to a crash or undefined behavior if done during an ongoing election with a special WAL write in progress.
- Fixed several crashes and/or undefined behaviors (assertions in debug build) which could appear when new synchronous transactions were made during ongoing elections (gh-6842).

### LuaJIT

- Fixed optimization for single-char strings in the `IR_BUFPUT` assembly routine.
- Fixed slots alignment in `lj-stack` command output when `LJ_GC64` is enabled (gh-5876).
- Fixed dummy frame unwinding in `lj-stack` command.
- Fixed top part of Lua stack (red zone, free slots, top slot) unwinding in `lj-stack` command.
- Added the value of `g->gc.mmudata` field to `lj-gc` output.
- Fixed detection of inconsistent renames even in the presence of sunk values (gh-4252, gh-5049, gh-5118).
- Fixed the order VM registers are allocated by LuaJIT frontend in case of `BC_ISGE` and `BC_ISGT` (gh-6227).
- Fixed inconsistency while searching for an error function when unwinding a C-protected frame to handle a runtime error (an error in `__gc` handler).
- `string.char()` builtin recording is fixed in case when no arguments are given (gh-6371, gh-6548).
- Actually made JIT respect `maxirconst` trace limit while recording (gh-6548).

### Lua

- Fixed a bug when multibyte characters broke `space:fselect()` output.
- When an error is raised during encoding call results, the auxiliary lightuserdata value is not removed from the main Lua coroutine stack. Prior to the fix, it leads to undefined behavior during the next usage of this Lua coroutine (gh-4617).
- Fixed Lua C API misuse, when the error is raised during call results encoding on unprotected coroutine and expected to be caught on the different one that is protected (gh-6248).
- Fixed `net.box` error in case connections are frequently opened and closed (gh-6217).
- Fixed incorrect handling of variable number of arguments in `box.func:call()` (gh-6405).

- Fixed `table.equals` result when booleans compared (gh-6386).
- Tap subtests inherit strict mode from parent (gh-6868).
- Fixed the behavior of Tarantool console on `SIGINT`. Now `Ctrl+C` discards the current input and prints the new prompt (gh-2717).

### Triggers

- Fixed the possibility of a crash in case when trigger removes itself.
- Fixed the possibility of a crash in case someone destroys trigger when it's yielding (gh-6266).

### SQL

- User-defined functions can now return `VARBINARY` to SQL as a result (gh-6024).
- Fixed assert on a cast of `DOUBLE` value greater than -1.0 and less than 0.0 to `INTEGER` and `UNSIGNED` (gh-6255).
- Removed spontaneous conversion from `INTEGER` to `DOUBLE` in a field of type `NUMBER` (gh-5335).
- All arithmetic operations can now only accept numeric values (gh-5756).
- Now function `quote()` returns an argument in case the argument is `DOUBLE`. The same for all other numeric types. For types other than numeric, `STRING` is returned (gh-6239).
- The `TRIM()` function now does not lose collation when executed with the keywords `BOTH`, `LEADING`, or `TRAILING` (gh-6299).
- Now getting unsupported msgpack extension in SQL throws the correct error (gh-6375).
- Now, when copying an empty string, an error will not be set unnecessarily (gh-6157, gh-6399).
- Fixed wrong comparison between `DECIMAL` and large `DOUBLE` values (gh-6376).
- Fixed truncation of `DECIMAL` during implicit cast to `INTEGER` in `LIMIT` and `OFFSET`.
- Fixed truncation of `DECIMAL` during implicit cast to `INTEGER` when value is used in an index.
- Fixed assert on a cast of `DECIMAL` value that is greater than -1.0 and less than 0.0 to `INTEGER` (gh-6485).
- The `HEX()` SQL built-in function no longer throws an assert when its argument consists of zero-bytes (gh-6113).
- `LIMIT` is now allowed in `ORDER BY` where sort order is in both directions (gh-6664).
- Fixed a memory leak in SQL during calling of user-defined function (gh-6789).
- Fixed assertion or segmentation fault when `MP_EXT` received via `net.box` (gh-6766).
- Now the `ROUND()` function properly supports `INTEGER` and `DECIMAL` as the first argument (gh-6988).
- Fixed a crash when a table inserted data into itself with an incorrect number of columns (gh-7132).

### Box

- Fixed `log.cfg` getting updated on `box.cfg` error (gh-6086).
- Fixed the error message in an attempt to insert into a tuple the size of which equals to `box.schema.FIELD_MAX` (gh-6198).
- We now check that all privileges passed to `box.schema.grant` are resolved (gh-6199).
- Added iterator type checking and allow passing iterator as a `box.index.{ALL,GT,...}` directly (gh-6501).

### Datetime

- Intervals received after datetime arithmetic operations may be improperly normalized if the result was negative

```
tarantool> date.now() - date.now()
---
- -1.000026000 seconds
...
```

  It means that two immediately called `date.now()` produce very close values, which difference should be close to 0, not 1 second (gh-6882).
- Fixed a bug in datetime module when `date:set{tzoffset=XXX}` did not produce the same result with `date.new{tzoffset=XXX}` for the same set of attributes passed (gh-6793).

### HTTP client

- Fixed invalid headers after redirect (gh-6101).

### Mvcc

- Fixed MVCC interaction with ephemeral spaces: TX manager now ignores such spaces (gh-6095).
- Fixed a loss of tuple after a conflict exception (gh-6132).
- Fixed a segmentation fault in update/delete of the same tuple (gh-6021).

### Net.box

- Changed the type of the error returned by `net.box` on timeout from ClientError to TimedOut (gh-6144).

### Recovery

- When `force_recovery` cfg option is set, Tarantool is able to boot from `snap`/`xlog` combinations where `xlog` covers changes committed both before and after `snap` creation. For example, `0...0.xlog`, covering everything up to `vclock {1: 15}` and `0...09.snap`, corresponding to `vclock {1: 9}` (gh-6794).

### Tarantoolctl

- Fixed the missing `rocks` keyword in `tarantoolctl rocks` help messages.

### Build

- Bumped Debian packages tarantool-common dependency to use luarocks 3 (gh-5429).
- Fixed an error when it was possible to have new Tarantool package (version >= 2.2.1) installed with pre-luarocks 3 tarantool-common package (version << 2.2.1), which caused rocks install to fail.
- The Debian package does not depend on binutils anymore (gh-6699).
- Fixed build errors with glibc-2.34 (gh-6686).
- Changed size of alt. signal stack for ASAN needs.
- Fixed build errors on arm64 with `CMAKE_BUILD_TYPE=Debug`.

## Tarantool 2.8.4

Released on 2022-04-25.

- Release: v. 2.8.4.
- Tag: `2.8.4`.

### Overview

2.8.4 is the third *stable* version of the 2.8 release series. It introduces 1 improvement and resolves 16 bugs since version *2.8.3*.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

#### Functionality added or changed

#### Build

- Support fedora-35 build (gh-6692).

#### Bugs fixed

#### Core

- Fixed a crash that could happen in case a tuple is deleted from a functional index while there is an iterator pointing to it (gh-6786).
- Fixed memory leak in interactive console (gh-6817).
- Fixed an assertion fail when passing tuple without primary key fields to before_replace trigger. Now tuple format is checked before execution of before_replace triggers and after each one (gh-6780).
- Banned DDL operations in space `on_replace` triggers, since they could lead to a crash (gh-6920).
- Fixed a bug due to which all fibers created with `fiber_attr_setstacksize()` leaked until the thread exit. Their stacks also leaked except when `fiber_set_joinable(..., true)` was used.

#### Vinyl

- Immediate removal of compacted run files created after the last checkpoint optimization now works for replica's initial JOIN stage (gh-6568).
- Fixed crash during recovery of a secondary index in case the primary index contains incompatible phantom tuples (gh-6778).

#### Raft

- Reconfiguration of `box.cfg.election_timeout` could lead to a crash or undefined behaviour if done during an ongoing election with a special WAL write in progress.

#### LuaJIT

- Fixed top part of Lua stack (red zone, free slots, top slot) unwinding in `lj-stack` command.
- Added the value of `g->gc.mmudata` field to `lj-gc` output.
- `string.char()` builtin recording is fixed in case when no arguments are given (gh-6371, gh-6548).
- Actually made JIT respect `maxirconst` trace limit while recording (gh-6548).

#### Lua

- Fixed `table.equals` result when booleans compared (gh-6386).
- Tap subtests inherit strict mode from parent (gh-6868).

#### Box

- Added iterator type checking and allow to pass iterator as a `box.index.{ALL,GT,...}` directly (gh-6501).

#### HTTP client

- Fixed invalid headers after redirect (gh-6101).

#### Recovery

- When `force_recovery` cfg option is set, Tarantool is able to boot from `snap`/`xlog` combinations where `xlog` covers changes committed both before and after `snap` creation. For example, `0...0.xlog`, covering everything up to `vclock {1: 15}` and `0...09.snap`, corresponding to `vclock {1: 9}` (gh-6794).

### Tarantool 2.8.3

Released on 2021-12-22.

- Release: v. 2.8.3.
- Tag: `2.8.3`.

#### Overview

2.8.3 is the third *stable* version of the 2.8 release series. It introduces 3 improvements and resolves 24 bugs since version *2.8.2*.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

#### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Functionality added or changed

#### LuaJIT

- Introduced support for `LJ_DUALNUM` mode in luajit-gdb.py (gh-6224).

#### Build

- Stop support of Ubuntu Trusty (14.04). (gh-6502)
- Bump debian package compatibility level to 10 (gh-5429). Bump minimal required debhelper to version 10 (except for Ubuntu Xenial).

### Bugs fixed

#### Core

- Disallow yields after DDL operations in MVCC mode. It fixes crash which takes place in case several transactions refer to system spaces (gh-5998).
- Fixed bug in MVCC connected which happens on rollback after DDL operation (gh-5998).
- Fix a bug when rollback resulted in unserializable behaviour (gh-6325)
- Fixed a crash during replace of malformed tuple into _schema system space (gh-6332).
- Fix memory leak in case of replace during background alter of primary index (gh-6290)
- Fix a bug when rollbacked changes appears in built-in-background index (gh-5958)
- Fix a bug when index was inconsistent after background build in case when the primary index is hash (gh-5977)
- Now inserting a tuple with the wrong "id" field into the _priv space will return the correct error (gh-6295).
- Fixed dirty read in MVCC after space alter (gh-6263, gh-6318).
- Fixed a crash caused by a race between box.session.push() and closing connection (gh-6520).
- Fixed crash in case a fiber changing box.cfg.listen is woken up (gh-6480).
- Fixed box.cfg.listen not reverted to the old address in case the new one is invalid (gh-6092).
- Fixed a bug because of which the garbage collector could remove an xlog file that is still in use (gh-6554).
- Fix crash during granting priveleges from guest (gh-5389).

### Replication

- Fixed replica reconnecting to a living master on any `box.cfg{replication=...}` change. Such reconnects could lead to replica failing to restore connection for `replication_timeout` seconds (gh-4669).

### LuaJIT

- Fixed the order VM registers are allocated by LuaJIT frontend in case of `BC_ISGE` and `BC_ISGT` (gh-6227).
- Fixed inconsistency while searching for an error function when unwinding a C protected frame to handle a runtime error (e.g. an error in __gc handler).

### Lua

- When error is raised during encoding call results, auxiliary lightuserdata value is not removed from the main Lua coroutine stack. Prior to the fix it leads to undefined behaviour during the next usage of this Lua coroutine (gh-4617).
- Fixed Lua C API misuse, when the error is raised during call results encoding on unprotected coroutine and expected to be catched on the different one, that is protected (gh-6248).
- Fixed net.box error in case connections are frequently opened and closed (gh-6217).
- Fixed incorrect handling of variable number of arguments in box.func:call() (gh-6405).

### Triggers

- Fixed possibility crash in case when trigger removes itself. Fixed possibility crash in case when someone destroy trigger, when it's yield (gh-6266).

### SQL

- Now, when copying an empty string, an error will not be set unnecessarily (gh-6157, gh-6399).

### Build

- The Debian package does not depend on binutils anymore (gh-6699).
- Fix build errors with glibc-2.34 (gh-6686).

### Tarantool 2.8.2

Released on 2021-08-19.

- Release: v. 2.8.2.
- Tag: `2.8.2-0-gfc96d10`.

#### Overview

2.8.2 is the second *stable* version of the 2.8 release series. It introduces 6 improvements and resolves 51 bugs since version *2.8.1*.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

#### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

#### Functionality added or changed

##### LuaJIT

- Introduced support for `LJ_DUALNUM` mode in `luajit-gdb.py` (gh-6224).

##### Lua

- Introduced the new method `table.equals`. It compares two tables by value with respect to the `__eq` metamethod.

##### Logging

- The `log` module now supports symbolic representation of log levels. Now it is possible to specify levels the same way as in the `box.cfg{}` call (gh-5882).

  For example, instead of

  ```
  require('log').cfg{level = 6}
  ```

  it is possible to use

  ```
  require('log').cfg{level = 'verbose'}
  ```

### SQL

- Descriptions of type mismatch error and inconsistent type error have become more informative (gh-6176).
- Removed explicit cast from `BOOLEAN` to numeric types and vice versa (gh-4770).

  For example, `CAST(FALSE AS INTEGER)` was `0` in version 2.8. Now it causes an error.
- Removed explicit cast from `VARBINARY` to numeric types and vice versa (gh-4772, gh-5852).
- Fixed a bug where a string that is not `NULL`-terminated could not be cast to `BOOLEAN`, even if the conversion would be successful according to the rules.

### Build

- Fedora 34 builds are now supported (gh-6074).
- Fedora 28 and 29 builds are no longer supported.

### Bugs fixed

### Core

- **[Breaking change]** `fiber.wakeup()` in Lua and `fiber_wakeup()` in C became NOP on the currently running fiber. Previously they allowed "ignoring" the next yield or sleep, which resulted in unexpected erroneous wake-ups. Calling these functions right before `fiber.create()` in Lua or `fiber_start()` in C could lead to a crash (in debug build) or undefined behaviour (in release build) (gh-6043).

  There was a single use case for the previous behaviour: rescheduling in the same event loop iteration, which is not the same as `fiber.sleep(0)` in Lua and `fiber_sleep(0)` in C. It could be done in the following way:

  in C:

  ```
  fiber_wakeup(fiber_self());
  fiber_yield();
  ```

  and in Lua:

  ```
  fiber.self():wakeup()
  fiber.yield()
  ```

  To get the same effect in C, one can now use `fiber_reschedule()`. In Lua, it is now impossible to reschedule the current fiber directly in the same event loop iteration. One can reschedule self through a second fiber, but it is strongly discouraged:

  ```
  -- do not use this code
  local self = fiber.self()
  fiber.new(function() self:wakeup() end)
  fiber.sleep(0)
  ```
- Fixed memory leak on `box.on_commit()` and `box.on_rollback()` (gh-6025).

- `fiber_join()` now checks if the argument is a joinable fiber. The absence of this check could lead to unpredictable results. Note that the change affects the C level only; in the Lua interface, `fiber:join()` protection is already enabled.
- Now Tarantool yields when it scans `.xlog` files for the latest applied vclock and finds the right place to start recovering from. It means that the instance becomes responsive right after the `box.cfg` call even if an empty `.xlog` was not created on the previous exit.

  This fix also prevents the relay from timing out when a freshly subscribed replica needs rows from the end of a relatively long (hundreds of MBs) `.xlog` file (gh-5979).
- The counter in `N rows processed` log messages no longer resets on each newly recovered `xlog`.
- Fixed a crash in JSON update on tuple/space, where the update included two or more operations that accessed fields in reversed order and these fields didn't exist. Example: `box.tuple.new({1}):update({{'=', 4, 4}, {'=', 3, 3}})` (gh-6069).
- Fixed invalid results of the `json` module's `encode` function when it was used from the Lua garbage collector. For example, this could happen in functions used as `ffi.gc()` (gh-6050).
- Added a check for user input of the number of iproto threads: value must be greater than zero and less than or equal to 1000 (gh-6005).
- Changing a listed address can no longer cause iproto threads to close the same socket several times.
- Tarantool now always removes the Unix socket correctly when it exits.
- Simultaneously updating a key in different transactions does not longer result in a MVCC crash (gh-6131).
- Fixed a bug where memtx MVCC crashed during reading uncommitted DDL (gh-5515).
- Fixed a bug where memtx MVCC crashed if an index was created in the transaction thread (gh-6137).
- Fixed a MVCC segmentation fault that arose when updating the entire space concurrently (gh-5892).
- Fixed a bug with failed assertion after a stress update of the same key (gh-6193).
- Fixed a crash where `box.snapshot` could be called during an incomplete transaction (gh-6229).
- Fixed console client connection failure in case of request timeout (gh-6249).
- Added a missing broadcast to `net.box.future:discard()` so that now fibers waiting for a request result wake up when the request is discarded (gh-6250).
- `box.info.uuid`, `box.info.cluster.uuid`, and `tostring(decimal)` with any decimal number in Lua could sometimes return garbage if there were `__gc` handlers in the user's code (gh-6259).
- Fixed an error message that appeared in a particular case during MVCC operation (gh-6247).
- Fixed a repeatable read violation after delete (gh-6206).
- Fixed a bug where the MVCC engine didn't track the `select{}` hash (gh-6040).
- Fixed a crash in MVCC after a drop of space with several indexes (gh-6274).
- Fixed a bug where the GC could leave tuples in secondary indexes (gh-6234).
- Disallow yields after DDL operations in MVCC mode. It fixes a crash that took place when several transactions referred to system spaces (gh-5998).
- Fixed a bug in MVCC that happened on rollback after a DDL operation (gh-5998).
- Fixed a bug where rollback resulted in unserializable behavior (gh-6325).

### Vinyl

- Fixed possible keys divergence during secondary index build, which might lead to missing tuples (gh-6045).
- Fixed the race between Vinyl garbage collection and compaction that resulted in a broken vylog and recovery failure (gh-5436).

### Replication

- Fixed the use after free in the relay thread when using elections (gh-6031).
- Fixed a possible crash when a synchronous transaction was followed by an asynchronous transaction right when its confirmation was being written (gh-6057).
- Fixed an error where a replica, while attempting to subscribe to a foreign cluster with a different replicaset UUID, didn't notice it is impossible and instead became stuck in an infinite retry loop printing a `TOO_EARLY_SUBSCRIBE` error (gh-6094).
- Fixed an error where a replica, while attempting to join a cluster with exclusively read-only replicas available, just booted its own replicaset, instead of failing or retrying. Now it fails with an error about the other nodes being read-only so they can't register the new replica (gh-5613).
- Fixed error reporting associated with transactions received from remote instances via replication. Any error raised while such a transaction was being applied was always reported as `Failed to write to disk` regardless of what really happened. Now the correct error is shown. For example, `Out of memory`, or `Transaction has been aborted by conflict`, and so on (gh-6027).
- Fixed replication occasionally stopping with `ER_INVALID_MSGPACK` when the replica is under high load (gh-4040).
- Fixed a cluster sometimes being unable to bootstrap if it contains nodes with `election_mode` set to `manual` or `voter` (gh-6018).
- Fixed a possible crash when `box.ctl.promote()` was called in a cluster with more than three instances. The crash happened in the debug build. In the release build, it could lead to undefined behaviour. It was likely to happen if a new node was added shortly before the promotion (gh-5430).
- Fixed a rare error appearing when MVCC (`box.cfg.memtx_use_mvcc_engine`) was enabled and more than one replica joined the cluster. The join could fail with the error `"ER_TUPLE_FOUND: Duplicate key exists in unique index 'primary' in space '_cluster'"`. The same could happen at the bootstrap of a cluster having more than three nodes (gh-5601).

### Raft

- Fixed a rare crash with leader election enabled (any mode except `off`), which could happen if a leader resigned from its role while another node was writing something elections-related to WAL. The crash was in the debug build, and in the release build it would lead to undefined behaviour (gh-6129).
- Fixed an error where a new replica in a Raft cluster tried to join from a follower instead of a leader and failed with the error `ER_READONLY` (gh-6127).

### LuaJIT

- Fixed optimization for single-char strings in the `IR_BUFPUT` assembly routine.
- Fixed slots alignment in the `lj-stack` command output when `LJ_GC64` is enabled (gh-5876).
- Fixed dummy frame unwinding in the `lj-stack` command.
- Fixed detection of inconsistent renames even in the presence of sunk values (gh-4252, gh-5049, gh-5118).
- Fixed the VM register allocation order provided by LuaJIT frontend in case of `BC_ISGE` and `BC_ISGT` (gh-6227).

### Lua

- When an error occurs during encoding call results, the auxiliary lightuserdata value is not removed from the main Lua coroutine stack. Before the fix, it led to undefined behaviour during the next usage of this Lua coroutine (gh-4617).
- Fixed a Lua C API misuse when the error is raised during call results encoding in an unprotected coroutine and expected to be caught in a different, protected coroutine (gh-6248).

### Triggers

- Fixed a possible crash in case trigger removes itself. Fixed a possible crash in case someone destroys a trigger when it yields (gh-6266).

### SQL

- User-defined functions can now return a VARBINARY result to SQL (gh-6024).
- Fixed assert when a DOUBLE value greater than -1.0 and less than 0.0 is cast to INTEGER and UNSIGNED (gh-6225).
- Removed spontaneous conversion from INTEGER to DOUBLE in a field of the NUMBER type (gh-5335).
- All arithmetic operations can now accept numeric values only (gh-5756).

### MVCC

- Fixed MVCC interaction with ephemeral spaces: TX manager now ignores them (gh-6095).
- Fixed loss of tuples after a conflict exception (gh-6132).
- Fixed a segfault during update/delete of the same tuple (gh-6021).

### Tarantool 2.8.1

Дата выхода: 2021-04-21.

- Версия: v. 2.8.1.
- Тег версии: `2.8.1-0-ge2a1ec0`.

#### Общие сведения

2.8.1 — *бета-версия*, входящая в серию 2.8.

Эта версия содержит 28 новых функций и 31 исправление по сравнению с версией *2.7.2*. Возможны ошибки в менее распространенных функциях. Если вы обнаружили проблему, сообщите о ней на GitHub.

Важные изменения:

- Теперь в Tarantool можно задавать несколько потоков (threads) IPROTO.
- Setting `box.cfg` options with *environment variables*.
- Введена новая функция `box.ctl.promote()` и механизм ручных выборов лидера.
- Улучшен профилировщик памяти Lua.

#### Совместимость

Любая версия Tarantool 2.x обратно совместима с версиями Tarantool 1.10.x. Это касается структур двоичных данных, клиент-серверного протокола и протокола репликации.

Чтобы получить доступ ко всем новым функциям серии 2.x, обновите Tarantool с помощью `box.schema.upgrade()`.

#### Обновления функциональности

#### Основные функции (Core)

- При определении индексируемых полей теперь можно использовать параметр `exclude_null`. Тогда в индексе не будут сохраняться кортежи, соответствующее поле которых имеет значение `null` (gh-4480).

  Например, в индекс, созданный методом `s:create_index('sk', {parts={{2, 'number', exclude_null=true}}})`, не войдут кортежи `{1, null}` и `{2, null}`, но могут войти кортежи `{null, 1}` и `{1, 1}`.

- В `box.cfg{}` добавлен параметр `slab_alloc_granularity`, позволяющий настроить гранулярность выделения памяти для аллокатора small. Значение `slab_alloc_granularity` должно быть степенью двойки, но не менее 4 (gh-5518).

- В предыдущих версиях триггеры Lua `on_shutdown` запускались последовательно. Теперь каждый триггер запускается в отдельном файбере. По умолчанию Tarantool в течение 3 секунд ожидает, когда завершится обработка триггеров. Пользователь может изменить этот временной промежуток с помощью новой функции `box.ctl.set_on_shutdown_timeout`.

  Когда время истечёт, Tarantool немедленно завершит работу, не дожидаясь окончания обработки оставшихся триггеров.

- С помощью нового API `on_shutdown` (gh-5723) разработчики модулей Tarantool могут указывать функции, которые будут вызываться при остановке Tarantool. Подробнее: *Function on_shutdown*.
- Введена очередь журнала упреждающей записи (WAL). Размер очереди в байтах определяется новым параметром конфигурации `wal_queue_max_size`. Его значение по умолчанию — 16 МБ. Параметр позволяет ограничить объем транзакций, которые реплика заносит в WAL. Проверка ограничения происходит каждый раз, когда транзакция с мастера заносится в WAL на реплике. Как только транзакция будет успешно записана, ее место в очереди освободится (gh-5536).
- Информацию о состоянии синхронной репликации теперь можно получить через интерфейс `box.info.synchro` (gh-5191).
- Теперь Tarantool позволяет *запускать несколько потоков (threads) IPROTO*. Это полезно в ситуациях, когда единственный поток IPROTO становится узким местом производительности (gh-5645).
- При операции обновления кортежей (update) пропуск полей не поддерживается. Пропущенные поля теперь заменяются нулевыми значениями (gh-3378).
- Новый модуль `box.lib` позволяет загружать и выполнять хранимые процедуры на языке C на узлах, доступных только для чтения (gh-4642).
- Configuration options in `box.cfg` can now be *defined with environment variables* (gh-5602).

  Применяются параметры конфигурации из следующих источников в порядке приоритета:

  – `box.cfg{}`;

  – переменные окружения;

  – параметры `tarantoolctl`;

  – значения по умолчанию.

### Репликация

- Добавлена функция `box.ctl.promote()`, введены ручные выборы лидера. Чтобы их включить, укажите параметр `election_mode='manual'` (gh-3055).

  Экземпляр, находящийся в ручном (`manual`) режиме выборов, большую часть времени имеет роль `voter`, но при вызове `box.ctl.promote()` может начать выборы и стать лидером. Если `election_mode ~= 'manual'`, метод `box.ctl.promote()` делает то же, что устаревший метод `box.ctl.clear_synchro_queue()`.

### LuaJIT

- Отчет профилировщика памяти LuaJIT стал удобнее для чтения (gh-5811). Теперь проще понять, в какой строке кода происходит событие. Пользователь видит имя исходного файла и строку, где выделяется память, а также количество событий и статистику памяти в байтах. Номер строки с определением функции не отображается.

  **Критическое изменение**: информация о строке, в которой определяется функция, теперь сохраняется в поле `linedefined` таблицы символов. Поле `name` теперь называется `source` в соответствии с Lua API для отладки.
- Ряд изменений в парсере профилировщика памяти:

  – Теперь отчет парсера включает информацию об изменениях в состоянии динамической памяти, возникших за период работы профилировщика (gh-5812).

  - Чтобы посмотреть только эти изменения, используйте параметр `--leak-only`.
  - Новый встроенный модуль `memprof.process` производит постобработку и агрегацию событий, связанных с выделением памяти.

Чтобы запустить профилировщик памяти, используйте следующую команду:

```
tarantool -e 'require("memprof")(arg)' - --leak-only /tmp/memprof.bin
```

### Инструменты

- Реализованы новые инструменты для анализа инцидентов и сбора соответствующих артефактов (gh-5569).

### Сборка

- Инфраструктура сборки Tarantool теперь требует использования CMake версии 3.2 или более поздней.
- Доступны бинарные пакеты для Fedora 33 (gh-5502).
- Начиная с этой версии, бинарные пакеты для CentOS 6 и Debian Jessie не публикуются.
- Среди зависимостей для RPM- и DEB-пакетов больше нет `autotools` (следствие gh-4968).
- Регулярное тестирование на MacOs 10.13 больше не проводится. Поддержка Tarantool для этой версии MacOS прекращена.
- Встроенный модуль `zstd` обновлён с версии 1.3.3 до версии 1.4.8 (в рамках gh-5502).
- Библиотека `libcurl`, входящая в сборку, теперь поддерживает протоколы SMTP и SMTPS (gh-4559).
- Файлы заголовков библиотеки `libcurl`, входящей в сборку, устанавливаются в системный каталог `${PREFIX}/include/tarantool` (gh-4559).

### Тестирование

- CI/CD Tarantool теперь проводится посредством GitHub Actions (gh-5662).
- Тестирование с помощью Jepsen на отдельных узлах теперь происходит каждый раз, когда выполняется push новых тегов, а также по cron каждые 3 часа (gh-5736).
- Фаззинг-тесты запускаются каждый раз, когда выполняется push (gh-1809).
- Реализовано независимое окружение для тестирования LuaJIT. Система сборки LuaJIT частично портирована на CMake. Инструменты для тестирования теперь находятся в репозитории tarantool/luajit (gh-4862, gh-5470).
- Теперь в инфраструктуре тестирования по умолчанию установлен Python 3 (gh-5652).

### Исправленные ошибки

### Основные функции (Core)

- Параметры индексируемых полей больше не пропускаются в случаях, когда тип поля не определён (gh-5674).
- Функция `lbox_ctl_is_recovery_finished()` больше не возвращает значение `true` в случае, если процесс восстановления ещё не завершён.
- Устранена ошибка повреждения памяти в модуле `net.box`. До этого исправления область памяти, занятая структурой `error`, освобождалась преждевременно, поскольку методы `ffi.gc` и `ffi.cast` вызывались не в том порядке.
- При передаче данных на реплику, которая присоединяется к кластеру или синхронизируется с мастером, больше не может истечь время ожидания (gh-5762).
- Устранена проблема, из-за которой модуль `net.box` получал схему индекса, в которой не содержится значение `path` (gh-5451).
- Раньше интенсивное использование модулей `uri` и `uuid` при уровне логирования DEBUG приводило к сбоям, а результаты выполнения функций, входящих в эти модули, могли быть повреждены. Теперь эта проблема решена. Та же проблема устранена для случаев, когда модули `uri` и `uuid` используются в callback-функциях, передаваемых в метод `ffi.gc()`, а также для некоторых функций, входящих в модули `fio`, `box.tuple` и `iconv` (gh-5632).
- Новый параметр `wal_cleanup_delay` предотвращает преждевременное удаление файлов `*.xlog`, необходимых репликам. Ранее при удалении возникала ошибка `XlogGapError` (gh-5806).
- Когда на мастере есть синхронные спейсы, во время стадии join на реплике больше не возникают ошибки `Unknown request type 40` (gh-5566).
- Исправлена ошибка при перезагрузке скомпилированного модуля, возникавшая, если обновлённый модуль не содержал части функций, которые были в прежнем коде. Раньше событие запускало процедуру восстановления удалённых функций. Однако вместо того, чтобы восстанавливать каждую функцию по отдельности, Tarantool ошибочно обрабатывал единственную запись, что приводило к сбою при вызове любой из этих функций (gh-5968).
- Решена проблема фантомных чтений: теперь механизм MVCC движка memtx отслеживает интервалы чтения (gh-5628).
- Функция `space:count()`, используемая с механизмом MVCC движка memtx, больше не выдаёт ошибочный результат (gh-5972).
- Исправлена проблема «грязного чтения» после перезапуска, возникающая при использовании механизма MVCC и синхронной репликации (gh-5973).

### Репликация

- Устранена проблема, из-за которой applier не обрабатывал сообщения `CONFIRM` или `ROLLBACK` от мастера и не давал ответа реплике.
- Устранена проблема, из-за которой некоторые кортежи не отправлялись с мастера на анонимную реплику, отставшую от него и пытающуюся зарегистрироваться в кластере.
- Исправлена ошибка, из-за которой синхронная транзакция подтверждалась и отображалась на реплике, а после перезагрузки исчезала вновь. Чаще всего это происходило в спейсах memtx при включённом механизме `memtx_use_mvcc_engine` (gh-5213).

- Устранены проблемы с восстановлением отмененной синхронной транзакции, состоящей из нескольких инструкций. Если такая транзакция относилась к асинхронным спейсам, она могла быть частично применена или восстановлена с ошибками (gh-5874).
- Исправлена ошибка синхронной репликации, из-за которой при подключении достаточно старого экземпляра отменённые транзакции могли быть применены вновь (gh-5445).

### Модуль swim

- Исправлена ошибка, из-за которой метод `<swim_instance>:broadcast()` не работал на нелокальных адресах и постоянно заносил в журнал сообщения об ошибке «Permission denied». Если работа экземпляра прекращалась, метод мог вернуть ненулевой код завершения даже при отсутствии ошибок в скрипте, а затем снова начать массово логировать сообщения о той же ошибке (gh-5864).
- Устранён сбой, возникавший при вызове `swim:member_by_uuid()` с аргументом `nil`/`box.NULL` или без аргументов (gh-5951).
- Устранён сбой, возникавший при попытке передать объект неподходящего типа в метод `__serialize` реплики-участника swim в Lua (gh-5952).

### LuaJIT

- Изменение размера стека Lua больше не приводит к ошибочному поведению профилировщика памяти (gh-5842).
- Исправлена ошибка, из-за которой значение `gc_cdatanum` в метриках платформы LuaJIT уменьшалось на две единицы. Ошибка возникала в случаях, когда финализатор был назначен для объекта типа `GCcdata` (gh-5820).

### Lua

- Устранена проблема с параметром `-e`. Ранее, если в качестве стандартного устройства ввода был установлен терминал, команда `tarantool` запускала интерактивный режим. Теперь команда `tarantool -e "print('Hello')")` просто выводит слово `Hello` и завершает работу (gh-5040).
- Устранена утечка кортежа, возникавшая при ошибке сериализации ключа во время выполнения `key_def:compare_with_key(кортеж, ключ)` (gh-5388).

### SQL

- Пользовательские C- или Lua-функции теперь получают ровно те же строки, что в них передаются посредством SQL. Различия могли возникать, если строка содержала `\\0` (gh-5938).
- Инструкции `SQL SELECT` и `SQL UPDATE` в отношении полей `UUID` и `DECIMAL` больше не вызывают ошибок сегментации (gh-5011, gh-5704, gh-5913).
- Устранена проблема, из-за которой инструкции `SELECT` и `GROUP BY`, выполняемые в рамках одной транзакции, приводили к неверным результатам. Проблема возникала, когда одно из обрабатываемых значений имело тип `VARBINARY`, а обращение к нему не происходило с помощью инструкции SELECT напрямую (gh-5890).

### Сборка

- Устранена проблема сборки на FreeBSD, связанная с неполным определением типа `struct sockaddr` (gh-5748).
- Уже скачанные зависимости статической сборки не будут скачиваться повторно (gh-5761).
- В ходе восстановления с помощью параметра `force_recovery` теперь удаляются файлы `.vylog`, созданные позднее, чем снимок данных. Благодаря этому экземпляр может восстанавливаться после инцидентов, возникающих во время сохранения контрольной точки (gh-5823).
- Устранены проблемы с конфигурацией `libcurl`, возникавшие, когда для сборки Tarantool использовалась команда `cmake3`, а в переменной `PATH` отсутствовал путь к `cmake` (gh-5955).

  Это исправление влияет на сборку Tarantool со встроенной библиотекой `libcurl` (стандартный тип сборки).

## Tarantool 2.7.3

Released on 2021-08-19.

- Release v. 2.7.3.
- Tagged `2.7.3-0-gdddf926`

### Overview

2.7.3 is the second *stable* version of the 2.7 release series. It introduces 6 improvements and resolves 49 bugs since version *2.7.2*.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Compatibility

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

Some changes are labeled as **[Breaking change]**. It means that the old behavior was considered error-prone and therefore changed to protect users from unintended mistakes. However, there is a small probability that someone can rely on the old behavior, and this label is to bring attention to the things that have been changed.

### Functionality added or changed

#### Core

- The information about the state of synchronous replication is now available via the `box.info.synchro` interface (gh-5191).

#### LuaJIT

- Introduced support for `LJ_DUALNUM` mode in `luajit-gdb.py` (gh-6224).

#### Lua

- Introduced the new method `table.equals`. It compares two tables by value with respect to the `__eq` metamethod.

#### SQL

- Descriptions of type mismatch error and inconsistent type error have become more informative (gh-6176).
- Removed explicit cast from `BOOLEAN` to numeric types and vice versa (gh-4770).

  For example, `CAST(FALSE AS INTEGER)` was `0` in version 2.8. Now it causes an error.
- Removed explicit cast from `VARBINARY` to numeric types and vice versa (gh-4772, gh-5852).
- Fixed a bug where a string that is not `NULL`-terminated could not be cast to `BOOLEAN`, even if the conversion would be successful according to the rules.

#### Build

- Fedora 34 builds are now supported (gh-6074).
- Fedora 28 and 29 builds are no longer supported.

### Bugs fixed

#### Core

- **[Breaking change]** `fiber.wakeup()` in Lua and `fiber_wakeup()` in C became NOP on the currently running fiber. Previously they allowed "ignoring" the next yield or sleep, which resulted in unexpected erroneous wake-ups. Calling these functions right before `fiber.create()` in Lua or `fiber_start()` in C could lead to a crash (in debug build) or undefined behaviour (in release build) (gh-6043).

  There was a single use case for the previous behaviour: rescheduling in the same event loop iteration, which is not the same as `fiber.sleep(0)` in Lua and `fiber_sleep(0)` in C. It could be done in the following way:

  in C:

```
fiber_wakeup(fiber_self());
fiber_yield();
```

and in Lua:

```
fiber.self():wakeup()
fiber.yield()
```

To get the same effect in C, one can now use `fiber_reschedule()`. In Lua, it is now impossible to reschedule the current fiber directly in the same event loop iteration. One can reschedule self through a second fiber, but it is strongly discouraged:

```
-- do not use this code
local self = fiber.self()
fiber.new(function() self:wakeup() end)
fiber.sleep(0)
```

- Fixed memory leak on `box.on_commit()` and `box.on_rollback()` (gh-6025).
- `fiber_join()` now checks if the argument is a joinable fiber. The absence of this check could lead to unpredictable results. Note that the change affects the C level only; in the Lua interface, `fiber:join()` protection is already enabled.
- Now Tarantool yields when it scans `.xlog` files for the latest applied vclock and finds the right place to start recovering from. It means that the instance becomes responsive right after the `box.cfg` call even if an empty `.xlog` was not created on the previous exit.

  This fix also prevents the relay from timing out when a freshly subscribed replica needs rows from the end of a relatively long (hundreds of MBs) `.xlog` file (gh-5979).
- The counter in `N rows processed` log messages no longer resets on each newly recovered `xlog`.
- Fixed invalid results of the `json` module's `encode` function when it was used from the Lua garbage collector. For example, this could happen in functions used as `ffi.gc()` (gh-6050).
- Simultaneously updating a key in different transactions does not longer result in a MVCC crash (gh-6131).
- Fixed a bug where memtx MVCC crashed during reading uncommitted DDL (gh-5515).
- Fixed a bug where memtx MVCC crashed if an index was created in the transaction thread (gh-6137).
- Fixed a MVCC segmentation fault that arose when updating the entire space concurrently (gh-5892).
- Fixed crash in case of reloading a compiled module when the new module lacks some functions present in the former code. In turn, this event triggers a fallback procedure where we restore old functions, but instead of restoring each function, we process a sole entry only, leading to the crash later when these restored functions are called (gh-5968).
- Fixed a bug with failed assertion after a stress update of the same key (gh-6193).
- Fixed a crash where `box.snapshot` could be called during an incomplete transaction (gh-6229).
- Fixed console client connection failure in case of request timeout (gh-6249).
- Added a missing broadcast to `net.box.future:discard()` so that now fibers waiting for a request result wake up when the request is discarded (gh-6250).
- `box.info.uuid`, `box.info.cluster.uuid`, and `tostring(decimal)` with any decimal number in Lua could sometimes return garbage if there were `__gc` handlers in the user's code (gh-6259).
- Fixed an error message that appeared in a particular case during MVCC operation (gh-6247).

- Fixed a repeatable read violation after delete (:gh-6206).
- Fixed a bug where the MVCC engine didn't track the `select{}` hash (gh-6040).
- Fixed a crash in MVCC after a drop of space with several indexes (gh-6274).
- Fixed a bug where the GC could leave tuples in secondary indexes (gh-6234).
- Disallow yields after DDL operations in MVCC mode. It fixes a crash that took place when several transactions referred to system spaces (gh-5998).
- Fixed a bug in MVCC that happened on rollback after a DDL operation (gh-5998).
- Fixed a bug where rollback resulted in unserializable behavior (gh-6325).

### Vinyl

- Fixed possible keys divergence during secondary index build, which might lead to missing tuples (gh-6045).
- Fixed the race between Vinyl garbage collection and compaction that resulted in a broken vylog and recovery failure (gh-5436).

### Replication

- Fixed the use after free in the relay thread when using elections (gh-6031).
- Fixed a possible crash when a synchronous transaction was followed by an asynchronous transaction right when its confirmation was being written (gh-6057).
- Fixed an error where a replica, while attempting to subscribe to a foreign cluster with a different replicaset UUID, didn't notice it is impossible and instead became stuck in an infinite retry loop printing a `TOO_EARLY_SUBSCRIBE` error (gh-6094).
- Fixed an error where a replica, while attempting to join a cluster with exclusively read-only replicas available, just booted its own replicaset, instead of failing or retrying. Now it fails with an error about the other nodes being read-only so they can't register the new replica (gh-5613).
- Fixed error reporting associated with transactions received from remote instances via replication. Any error raised while such a transaction was being applied was always reported as `Failed to write to disk` regardless of what really happened. Now the correct error is shown. For example, `Out of memory`, or `Transaction has been aborted by conflict`, and so on (gh-6027).
- Fixed replication occasionally stopping with `ER_INVALID_MSGPACK` when the replica is under high load (gh-4040).
- Fixed a cluster sometimes being unable to bootstrap if it contains nodes with `election_mode` set to `manual` or `voter` (gh-6018).
- Fixed a possible crash when `box.ctl.promote()` was called in a cluster with more than three instances. The crash happened in the debug build. In the release build, it could lead to undefined behaviour. It was likely to happen if a new node was added shortly before the promotion (gh-5430).
- Fixed a rare error appearing when MVCC (`box.cfg.memtx_use_mvcc_engine`) was enabled and more than one replica joined the cluster. The join could fail with the error `"ER_TUPLE_FOUND: Duplicate key exists in unique index 'primary' in space '_cluster'"`. The same could happen at the bootstrap of a cluster having more than three nodes (gh-5601).

### Raft

- Fixed a rare crash with leader election enabled (any mode except `off`), which could happen if a leader resigned from its role while another node was writing something elections-related to WAL. The crash was in the debug build, and in the release build it would lead to undefined behaviour (gh-6129).
- Fixed an error where a new replica in a Raft cluster tried to join from a follower instead of a leader and failed with the error `ER_READONLY` (gh-6127).

### LuaJIT

- Fixed optimization for single-char strings in the `IR_BUFPUT` assembly routine.
- Fixed slots alignment in the `lj-stack` command output when `LJ_GC64` is enabled (gh-5876).
- Fixed dummy frame unwinding in the `lj-stack` command.
- Fixed detection of inconsistent renames even in the presence of sunk values (gh-4252, gh-5049, gh-5118).
- Fixed the VM register allocation order provided by LuaJIT frontend in case of `BC_ISGE` and `BC_ISGT` (gh-6227).

### Lua

- When an error occurs during encoding call results, the auxiliary lightuserdata value is not removed from the main Lua coroutine stack. Before the fix, it led to undefined behaviour during the next usage of this Lua coroutine (gh-4617).
- Fixed a Lua C API misuse when the error is raised during call results encoding in an unprotected coroutine and expected to be caught in a different, protected coroutine (gh-6248).

### Triggers

- Fixed a possible crash in case trigger removes itself. Fixed a possible crash in case someone destroys a trigger when it yields (gh-6266).

### SQL

- User-defined functions can now return a VARBINARY result to SQL (gh-6024).
- Fixed assert when a DOUBLE value greater than -1.0 and less than 0.0 is cast to INTEGER and UNSIGNED (gh-6225).
- Removed spontaneous conversion from INTEGER to DOUBLE in a field of the NUMBER type (:gh-5335).
- All arithmetic operations can now accept numeric values only (gh-5756).

### MVCC

- Fixed MVCC interaction with ephemeral spaces: TX manager now ignores them (gh-6095).
- Fixed loss of tuples after a conflict exception (gh-6132).
- Fixed a segfault during update/delete of the same tuple (gh-6021).

## Tarantool 2.7.2

Released on 2021-04-21.

- Release v. 2.7.2.
- Tagged `2.7.2-0-g4d8c068`

### Общие сведения

2.7.2 is the first *stable* version of the 2.7 release series. It introduces 15 improvements and resolves 30 bugs since version *2.7.1*.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Изменения или добавления функциональности

### Core

- Introduced the concept of WAL queue and the new configuration option `wal_queue_max_size`, measured in bytes. The default value is 16 Mb. The option helps limit the pace at which replica submits new transactions to the WAL. The limit is checked every time a transaction from the master is submitted to the replica's WAL. The space taken by the transaction is considered empty once it's successfully written (gh-5536).

### Репликация

- Introduced the `box.ctl.promote()` function and the concept of manual elections (enabled with `election_mode='manual'`) (gh-3055).

  Once the instance is in the `manual` election mode, it acts like a `voter` most of the time, but may trigger elections and become a leader when `box.ctl.promote()` is called. When `election_mode ~= 'manual'`, `box.ctl.promote()` replaces `box.ctl.clear_synchro_queue()`, which is now deprecated.

### Build

- Tarantool build infrastructure now requires CMake version 3.1 or later.
- Binary packages for Fedora 33 are now available (gh-5502) .
- Binary packages for CentOS 6 and Debian Jessie won't be published since this version.
- RPM and DEB packages no longer have the `autotools` dependency (follows up gh-4968).
- Regular testing on MacOS 10.13 has been disabled, effectively stopping the support of this version.
- The built-in `zstd` is upgraded from v1.3.3 to v1.4.8 (part of gh-5502).
- SMTP and SMTPS protocols are now enabled in the bundled `libcurl` (gh-4559).
- The `libcurl` headers are now shipped to system path `${PREFIX}/include/tarantool` when `libcurl` is included as a bundled library or in a static build (gh-4559).

### Тестирование

- Tarantool CI/CD has migrated to GitHub Actions (gh-5662).
- Single node Jepsen testing now runs on per-push basis (gh-5736).
- A self-sufficient LuaJIT testing environment has been implemented. As a result, LuaJIT build system is now partially ported to CMake and all testing machinery is enclosed within the tarantool/luajit repository (gh-4862, gh-5470).
- Python 3 is now the default in the test infrastructure (gh-5652).

### Исправленные ошибки

### Core

- The index part options are no longer skipped when the field type is not specified (gh-5674).
- The `lbox_ctl_is_recovery_finished()` function no longer returns `true` when recovery is still in progress.
- A memory corruption bug has been fixed in netbox. The memory of a struct error which is still used will no longer be freed prematurely because of the wrong order of `ffi.gc` and `ffi.cast` calls.
- Relay can no longer time out while a replica is joining or syncing with the master. (gh-5762).
- An issue with missing «path» value of index schema fetched by netbox has been fixed (gh-5451).
- Extensive usage of `uri` and `uuid` modules with debug log level no longer leads to crashes or corrupted results of the functions from these modules. Same problem is resolved for using these modules from the callbacks passed to `ffi.gc()`, and for some functions from the modules `fio`, `box.tuple`, and `iconv` (gh-5632).
- The new `wal_cleanup_delay` option can prevent early cleanup of `*.xlog` files, needed by replicas. Such cleanup used to result in a `XlogGapError` (gh-5806).
- Appliers will no longer cause errors with `Unknown request type 40` during a final join when the master has synchronous spaces (gh-5566).
- Added memtx MVCC tracking of read gaps which fixes the problem of phantom reads (gh-5628).
- Fixed the wrong result of using `space:count()` with memtx MVCC (gh-5972).

- Fixed the dirty read after restart while using MVCC with synchronous replication (gh-5973).

### Репликация

- Fixed an issue with an applier hanging on a replica after failing to process a `CONFIRM` or `ROLLBACK` message coming from a master.
- Fixed the issue where master did not send some rows to an anonymous replica which had fallen behind and was trying to register.
- Fixed the bug when a synchronous transaction could be confirmed and visible on a replica, but then not confirmed or invisible again after restart. It was more likely to happen on memtx spaces with `memtx_use_mvcc_engine` enabled (gh-5213).
- Fixed the recovery of a rolled back multi-statement synchronous transaction which could lead to the transaction being applied partially, and to recovery errors. It happened in case the transaction worked with non-sync spaces (gh-5874).
- Fixed a bug in synchronous replication when rolled back transactions could reappear after reconnecting a sufficiently old instance (gh-5445).

### Module swim

- Fixed an issue where `<swim_instance>:broadcast()` did not work on non-local addresses and spammed «Permission denied» errors to the log. After instance termination it could return a non-0 exit code even if there were no errors in the script, and then spam the same error again (gh-5864).
- Fixed the crash on attempts to call `swim:member_by_uuid()` with no arguments or with `nil`/`box.NULL` (gh-5951).
- Fixed the crash on attempts to pass an object of a wrong type to `__serialize` method of a swim member in Lua (gh-5952).

### LuaJIT

- Lua stack resizing no longer results in a wrong behaviour of the memory profiler (gh-5842).
- Fixed a double `gc_cdatanum` decrementing in LuaJIT platform metrics which occurred when a finalizer was set for a `GCcdata` object (gh-5820).

### Lua

- Fixed the `-e` option, when `tarantool` used to enter the interactive mode when stdin is a TTY. Now, `tarantool -e 'print"Hello"'` doesn't enter the interactive mode, but just prints «Hello» and exits (gh-5040).
- Fixed a leak of a tuple object in `key_def:compare_with_key(tuple, key)`, which had occurred when the serialization of the key failed (gh-5388).

### SQL

- The string received by a user-defined C or Lua function will no longer be different from the string passed to the function. This could happen when the string passed from SQL had contained `\\0` (gh-5938).
- `SQL SELECT` or `SQL UPDATE` on `UUID` or `DECIMAL` field will not cause a SEGMENTATION FAULT anymore (gh-5011, gh-5704, gh-5913).
- Fixed an issue with wrong results of `SELECT` with `GROUP BY` which occurred when one of the selected values was `VARBINARY` and not directly obtained from the space (gh-5890).

### Build

- Fix building on FreeBSD (incomplete definition of type `struct sockaddr`) (gh-5748).
- The already downloaded static build dependencies will not be fetched repeatedly (gh-5761).
- Recovering with `force_recovery` option now deletes vylog files which are newer than the snapshot. It helps an instance recover after incidents during a checkpoint (gh-5823).
- Fixed the `libcurl` configuring when Tarantool itself has been configured with `cmake3` command and there was no `cmake` command in the `PATH` (gh-5955).

  This affects building Tarantool from sources with bundled `libcurl` (it is the default mode).

## Tarantool 2.7.1

Release: v. 2.7.1 Date: 2020-12-30 Tag: 2.7.1-0-g3ac498c9f

### Общие сведения

2.7.1 is the *beta* version of the 2.7 release series.

This release introduces 12 new features and resolves 21 bugs since the 2.6.1 version. There can be bugs in less common areas. If you find any, feel free to report an issue on GitHub.

Notable changes are:

- *LuaJIT memory profiler*;
- Expression evaluation for `replication_synchro_quorum`;
- The *ALTER TABLE ADD COLUMN* statement.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Изменения или добавления функциональности

#### Core

- Now it is allowed to define an index without extra braces when there is only one part: `parts = {field1, type1, ...}` (gh-2866). Read more in the *note about index parts declaration*.
- *Bitset index* now supports the varbinary type (gh-5392).
- Index-related options now can't be specified in their definition due to a more pedantic key-parts verification (gh-5473).
- A warning is now logged when schema version is older than last available schema version (gh-4574).
- UUID values created via `uuid` Lua module (`require('uuid')`) can now be compared using the comparison operators like `<`, `>=`, and others (gh-5511).
- The new `box.ctl.is_recovery_finished()` function allows user to determine whether memtx recovery is finished.
- The `force_recovery` option now ignores errors during snapshot recovery (gh-5422).
- Feedback daemon now reports `box.stat().*.total`, `box.stat.net().*.total`, and `box.stat.net().*.current` together with the time of report generation. The added fields reside in `feedback.stats.box`, `feedback.stats.net`, and `feedback.stats.time` (gh-5589).

#### LuaJIT

- Introduced the LuaJIT memory profiler (gh-5442) and the profile parser (gh-5490). Read more: *LuaJIT memory profiler*.

#### Lua

- Show JSON tokens themselves instead of token names `T_*` in the JSON decoder error messages (gh-4339).
- Show a decoding context in the JSON decoder error messages (gh-4339).

#### SQL

- The *ALTER TABLE ADD COLUMN* statement is now supported for empty tables (gh-2349, gh-3075).

#### Репликация

- It is now possible to specify synchro quorum as a function of a number `N` of registered replicas instead of a `const` number, for example:

```
box.cfg{replication_synchro_quorum = "N/2 + 1"}
```

  Only the non-anonymous bootstrapped replicas amount to `N`. The expression should respect synchro guarantees: at least 50% of the cluster size + 1. The expression value is re-evaluated automatically inside of Tarantool when new replicas appear or old ones are removed (gh-5446).

### Build

- Deploy packages for Fedora 32 (gh-4966).
- Deploy packages for Debian Bullseye (gh-5638).

### Разное

- If Tarantool crashes, it will now send a crash dump report to the feedback server. This report contains some fields from `uname` output, build information, crash reason, and a stack trace. You can disable crash reporting with `box.cfg{feedback_crashinfo = false}` (gh-5261).

### Исправленные ошибки

### Core

- `fiber.cond:wait()` now correctly throws an error when a fiber is cancelled, instead of ignoring the timeout and returning without any signs of an error (gh-5013).
- Fixed a memory corruption issue, which was most visible on macOS, but could affect any system (gh-5312).
- A dynamic module now gets correctly unloaded from memory in case of an attempt to load a non-existing function from it (gh-5475).
- A `swim:quit()` call now can't result in a crash (gh-4570).
- Snapshot recovery with no JSONPath or multikey indices involved now has normal performance (gh-4774).

### Репликация

- A false-positive "too long WAL write" message no longer appears for synchronous transactions (gh-5139).
- A `box.ctl.wait_rw()` call could return when the instance was not in fact writable due to having foreign synchronous transactions. As a result, there was no proper way to wait until the automatically elected leader would become writable. Now `box.ctl.wait_rw()` works correctly (gh-5440).
- Fixed a couple of crashes on various tweaks of election mode (gh-5506).
- Now `box.ctl.clear_synchro_queue` tries to commit everything that is present on the node. In order to do so it waits for other instances to replicate the data for `replication_synchro_quorum` seconds. In case timeout passes and quorum wasn't reached, nothing is rolled back (gh-5435).

### SQL

- Data changes in read-only mode are now forbidden (gh-5231).
- Query execution now does not occasionally raise an unrelated error "Space '0' does not exist" (gh-5592).
- Coinciding names of temporary files (used to store data during execution) having two instances running on the same machine no longer cause a segfault (gh-5537).
- The return value of `ifnull()` built-in function is now of a correct type.
- SQL calling Lua functions with box calls inside can no longer result in a memory corruption (gh-5427).

### LuaJIT

- Dispatching `__call` metamethod no longer causes address clashing (gh-4518, gh-4649).
- Fixed a false positive panic when yielding in debug hook (gh-5649).

### Lua

- An attempt to use a `net.box` connection which is not established yet now results in a correctly reported error (gh-4787).
- Fixed a `NULL` dereference on error paths in merger which usually happened on a 'wrong' `key_def` (gh-5450).
- Calling `key_def.compare_with_key()` with an invalid key no longer causes a segfault (gh-5307).
- Fixed a hang which occured when `tarantool` ran a user script with the `-e` option and this script exited with an error (like with `tarantool -e 'assert(false)'`) (gh-4983).

### Memtx

- The `on_schema_init` triggers now can't cause duplicates in primary key (gh-5304).

### Tarantool 2.6.3

Released on 2021-04-21.

- Release v. 2.6.3.
- Tagged `2.6.3-0-gcd487a2`

### Общие сведения

2.6.3 is the second *stable* version of the 2.6 release series. It introduces 15 improvements and resolves 28 bugs since version *2.6.2*.

The «stable» label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Изменения или добавления функциональности

#### Core

- Introduced the concept of WAL queue and the new configuration option `wal_queue_max_size`, measured in bytes. The default value is 16 Mb. The option helps limit the pace at which replica submits new transactions to the WAL. The limit is checked every time a transaction from the master is submitted to the replica's WAL. The space taken by the transaction is considered empty once it's successfully written (gh-5536).

#### Репликация

- Introduced the `box.ctl.promote()` function and the concept of manual elections (enabled with `election_mode='manual'`) (gh-3055).

  Once the instance is in the `manual` election mode, it acts like a `voter` most of the time, but may trigger elections and become a leader when `box.ctl.promote()` is called. When `election_mode ~= 'manual'`, `box.ctl.promote()` replaces `box.ctl.clear_synchro_queue()`, which is now deprecated.

#### Build

- Tarantool build infrastructure now requires CMake version 3.1 or later.
- Binary packages for Fedora 33 are now available (gh-5502) .
- Binary packages for CentOS 6 and Debian Jessie won't be published since this version.
- RPM and DEB packages no longer have the `autotools` dependency (follows up gh-4968).
- Regular testing on MacOS 10.13 has been disabled, effectively stopping the support of this version.
- The built-in `zstd` is upgraded from v1.3.3 to v1.4.8 (part of gh-5502).
- SMTP and SMTPS protocols are now enabled in the bundled `libcurl` (gh-4559).
- The `libcurl` headers are now shipped to system path `${PREFIX}/include/tarantool` when `libcurl` is included as a bundled library or in a static build (gh-4559).

### Тестирование

- Tarantool CI/CD has migrated to GitHub Actions (gh-5662).
- Single node Jepsen testing now runs on per-push basis (gh-5736).
- A self-sufficient LuaJIT testing environment has been implemented. As a result, LuaJIT build system is now partially ported to CMake and all testing machinery is enclosed within the tarantool/luajit repository (gh-4862, gh-5470).
- Python 3 is now the default in the test infrastructure (gh-5652).

### Исправленные ошибки

#### Core

- The `lbox_ctl_is_recovery_finished()` function no longer returns `true` when recovery is still in progress.
- A memory corruption bug has been fixed in netbox. The memory of a struct error which is still used will no longer be freed prematurely because of the wrong order of `ffi.gc` and `ffi.cast` calls.
- Relay can no longer time out while a replica is joining or syncing with the master. (gh-5762).
- An issue with missing «path» value of index schema fetched by netbox has been fixed (gh-5451).
- Extensive usage of `uri` and `uuid` modules with debug log level no longer leads to crashes or corrupted results of the functions from these modules. Same problem is resolved for using these modules from the callbacks passed to `ffi.gc()`, and for some functions from the modules `fio`, `box.tuple`, and `iconv` (gh-5632).
- The new `wal_cleanup_delay` option can prevent early cleanup of `*.xlog` files, needed by replicas. Such cleanup used to result in a `XlogGapError` (gh-5806).
- Appliers will no longer cause errors with `Unknown request type 40` during a final join when the master has synchronous spaces (gh-5566).
- Added memtx MVCC tracking of read gaps which fixes the problem of phantom reads (gh-5628).
- Fixed the wrong result of using `space:count()` with memtx MVCC (gh-5972).
- Fixed the dirty read after restart while using MVCC with synchronous replication (gh-5973).

#### Репликация

- Fixed an issue with an applier hanging on a replica after failing to process a `CONFIRM` or `ROLLBACK` message coming from a master.
- Fixed the issue where master did not send some rows to an anonymous replica which had fallen behind and was trying to register.
- Fixed the bug when a synchronous transaction could be confirmed and visible on a replica, but then not confirmed or invisible again after restart. It was more likely to happen on memtx spaces with `memtx_use_mvcc_engine` enabled (gh-5213).
- Fixed the recovery of a rolled back multi-statement synchronous transaction which could lead to the transaction being applied partially, and to recovery errors. It happened in case the transaction worked with non-sync spaces (gh-5874).

- Fixed a bug in synchronous replication when rolled back transactions could reappear after reconnecting a sufficiently old instance (gh-5445).

### Swim

- Fixed an issue where `<swim_instance>:broadcast()` did not work on non-local addresses and spammed «Permission denied» errors to the log. After instance termination it could return a non-0 exit code even if there were no errors in the script, and then spam the same error again (gh-5864).
- Fixed the crash on attempts to call `swim:member_by_uuid()` with no arguments or with `nil`/`box.NULL` (gh-5951).
- Fixed the crash on attempts to pass an object of a wrong type to `__serialize` method of a swim member in Lua (gh-5952).

### Lua

- Fixed the `-e` option, when `tarantool` used to enter the interactive mode when stdin is a TTY. Now, `tarantool -e 'print"Hello"'` doesn't enter the interactive mode, but just prints «Hello» and exits (gh-5040).
- Fixed a leak of a tuple object in `key_def:compare_with_key(tuple, key)`, which had occurred when the serialization of the key failed (gh-5388).

### SQL

- The string received by a user-defined C or Lua function will no longer be different from the string passed to the function. This could happen when the string passed from SQL had contained `\\0` (gh-5938).
- `SQL SELECT` or `SQL UPDATE` on `UUID` or `DECIMAL` field will not cause a SEGMENTATION FAULT anymore (gh-5011, gh-5704, gh-5913).
- Fixed an issue with wrong results of `SELECT` with `GROUP BY` which occurred when one of the selected values was `VARBINARY` and not directly obtained from the space (gh-5890).

### LuaJIT

- Fixed a double `gc_cdatanum` decrementing in LuaJIT platform metrics which occurred when a finalizer was set for a `GCcdata` object (gh-5820).

### Build

- Fix building on FreeBSD (incomplete definition of type `struct sockaddr`) (gh-5748).
- The already downloaded static build dependencies will not be fetched repeatedly (gh-5761).
- Recovering with `force_recovery` option now deletes vylog files which are newer than the snapshot. It helps an instance recover after incidents during a checkpoint (gh-5823).
- Fixed the `libcurl` configuring when Tarantool itself has been configured with `cmake3` command and there was no `cmake` command in the `PATH` (gh-5955).

  This affects building Tarantool from sources with bundled `libcurl` (it is the default mode).

### Tarantool 2.6.2

Release: v. 2.6.2 Date: 2020-12-30 Tag: 2.6.2-0-g34d504d

#### Общие сведения

2.6.2 is the first *stable* version of the 2.6 release series. It introduces one improvement and resolves 21 bugs since 2.6.1.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

#### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

#### Изменения или добавления функциональности

##### Репликация

- It is now possible to specify synchro quorum as a function of a number `N` of registered replicas instead of a `const` number, for example:

  ```
  box.cfg{replication_synchro_quorum = "N/2 + 1"}
  ```

  Only the non-anonymous bootstrapped replicas amount to `N`. The expression should respect synchro guarantees: at least 50% of the cluster size + 1. The expression value is re-evaluated automatically inside of Tarantool when new replicas appear or old ones are removed (gh-5446).

##### Lua

- Show JSON tokens themselves instead of token names `T_*` in the JSON decoder error messages (gh-4339).
- Show a decoding context in the JSON decoder error messages (gh-4339).

##### Build

- Deploy packages for Fedora 32 (gh-4966).
- Deploy packages for Debian Bullseye (gh-5638).

### Исправленные ошибки

#### Core

- `fiber.cond:wait()` now correctly throws an error when a fiber is cancelled, instead of ignoring the timeout and returning without any signs of an error (gh-5013).
- Fixed a memory corruption issue, which was most visible on macOS, but could affect any system (gh-5312).
- A dynamic module now gets correctly unloaded from memory in case of an attempt to load a non-existing function from it (gh-5475).
- A `swim:quit()` call now can't result in a crash (gh-4570).
- Snapshot recovery with no JSONPath or multikey indices involved now has normal performance (gh-4774).

#### Репликация

- A false-positive "too long WAL write" message no longer appears for synchronous transactions (gh-5139).
- A `box.ctl.wait_rw()` call could return when the instance was not in fact writable due to having foreign synchronous transactions. As a result, there was no proper way to wait until the automatically elected leader would become writable. Now `box.ctl.wait_rw()` works correctly (gh-5440).
- Fixed a couple of crashes on various tweaks of election mode (gh-5506).
- Now `box.ctl.clear_synchro_queue` tries to commit everything that is present on the node. In order to do so it waits for other instances to replicate the data for `replication_synchro_quorum` seconds. In case timeout passes and quorum was not reached, nothing is rolled back (gh-5435).

#### SQL

- Data changes in read-only mode are now forbidden (gh-5231).
- Query execution now does not occasionally raise an unrelated error "Space '0' does not exist" (gh-5592).
- Coinciding names of temporary files (used to store data during execution) having two instances running on the same machine no longer cause a segfault (gh-5537).
- The return value of `ifnull()` built-in function is now of a correct type.
- SQL calling Lua functions with box calls inside can no longer result in a memory corruption (gh-5427).

#### LuaJIT

- Dispatching `__call` metamethod no longer causes address clashing (gh-4518, gh-4649).
- Fixed a false positive panic when yielding in debug hook (gh-5649).

### Lua

- An attempt to use a `net.box` connection which is not established yet now results in a correctly reported error (gh-4787).
- Fixed a `NULL` dereference on error paths in merger which usually happened on a 'wrong' `key_def` (gh-5450).
- Calling `key_def.compare_with_key()` with an invalid key no longer causes a segfault (gh-5307).
- Fixed a hang which occured when `tarantool` ran a user script with the `-e` option and this script exited with an error (like with `tarantool -e 'assert(false)'`) (gh-4983).

### Memtx

- The `on_schema_init` triggers now can't cause duplicates in primary key (gh-5304).

## Tarantool 2.6.1

Release: v. 2.6.1 Date: 2020-10-22 Tag: 2.6.1-0-gcfe0d1a

### Общие сведения

2.6.1 is the beta version of the 2.6 release series.

This release introduces roughly 17 features and resolves 22 bugs since the 2.5.1 version. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

Notable changes are:

- Transactional manager for the memtx engine that allows yielding in transactions. It also guarantees the data consistency for synchronous replication, eliminating dirty reads for unconfirmed transactions.

  Read more:

  - *Transaction mode: MVCC*
  - *use_mvcc_engine configuration parameter*

- Raft-based automated failover mechanism for a single-leader replica set. Replica set can be configured to provide automated leader election and failover. Accompanied with synchronous replication it brings data safety and service reliability on a new level.

  Read more:

  - *Automated leader election*
  - *election_mode configuration parameter*
  - *election_timeout configuration parameter*
  - *box.info.election*

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

There are changes labeled with **[Breaking change]**. It means that the old behavior was considered error-prone and therefore changed to protect users from unintended mistakes. However, there is a little probability that someone can lean on the old behavior, and this label is to bring attention to the things that have been changed.

### Изменения или добавления функциональности

#### Core

- Introduce a function to check that the specified value is of UUID type (gh-5171). Read more: *uuid.is_uuid()*.
- A new function `space:alter(options)` to change some space settings without recreation nor touching `_space` space. Read more: *space_object:alter()*.
- Composite types extraction is now supported in key_def (gh-4538).
- Make memtx tuple hints an optional feature (gh-4927). Read more: *Options for space_object:create_index()*.

#### Vinyl

Rework upsert operation in vinyl so that now (gh-5107):

- if upsert can't be applied it is skipped and corresponding error is logged (gh-1622);
- upserts now follow associative property: the result of several upserts does not depend on the order of their application (gh-5105);
- upserts referring to -1 fieldno are handled correctly now (gh-5087).
- there's no more upserts squash procedure: upserts referring to the same field with arithmetic operations are not merged into one operation since resulting upsert might not be applied - as a result both upserts would be ignored (meanwhile only one should be).

#### LuaJIT

- **[Breaking change]** Introduce LuaJIT platform metrics (gh-5187). Read more: *LuaJIT metrics*.
  - This change introduces new builtin library "misc" that may conflict with user's modules.

### SQL

- SQL views are not alterable anymore. Beforehand it led to the undefined behaviour.
- Introduce “automatic index” optimization. Ephemeral space with single index can be created to store and speed-up intermediate results access during query execution (gh-4933).

### Репликация

- Automated leader election based on Raft algorithm (gh-1146). Read more: *Automated leader election*.
- When election is enabled, a newly elected leader will automatically finish all the synchronous transactions, created by the old leader (gh-5339).

### Build

- Tarantool static build is enhanced in scope of gh-5095. It can be built on the host machine with no Docker at all. As a result it can be built using the OSX environment.

### Разное

- Add all exported symbols from bundled libcurl library (gh-5223)
- Add `fselect` method that is similar to select, but formats results like mysql would (gh-5161).

### Module API

- Exposed the box region, key_def and several other functions in order to implement external tuple.keydef and tuple.merger modules on top of them (gh-5273, gh-5384).

### Исправленные ошибки

### Core

- Fixed a bug related to ignoring internal getaddrinfo errors on macOS in logger (gh-4138).
- Fixed a crash when JSON tuple field access was used to get a multikey indexed field, and when a JSON contained [*] in the beginning (gh-5224).
- Fixed msgpack extension types decoding error message (gh-5016).
- Dropped restrictions on nullable multikey index root. They were introduced due to inaccuracy in multikey index realization. It is now fixed. Also all fields are now nullable by default as it was before 2.2.1 (gh-5192).
- Fixed fibers switch-over to prevent JIT machinery misbehavior. Trace recording is aborted when fiber yields the execution. The yield occurring while the compiled code is being run (it’s likely a function with a yield underneath called via LuaJIT FFI) leads to the platform panic (gh-1700, gh-4491).
- Fixed fibers switch-over to prevent implicit GC disabling. The yield occurring while user-defined __gc metamethod is running leads to the platform panic.

### Репликация

- Fixed a bug when a rolled back synchronous transaction could become committed after restart (gh-5140).
- Fixed crash in synchronous replication when master's local WAL write fails (gh-5146).
- Instance will terminate if a synchronous transaction confirmation or rollback fail. Before it was undefined behavior (gh-5159).
- Snapshot could contain changes from a rolled back synchronous transaction (gh-5167).
- Fixed a crash when synchronous transaction's rollback and confirm could be written simultaneously for the same LSN (gh-5185).
- Fixed a crash when replica cleared synchronous transaction queue, while it was not empty on master (gh-5195).
- During recovery of synchronous changes from snapshot the instance could crash (gh-5288).
- Having synchronous rows in the snapshot could make the instance hang on recovery (gh-5298).
- Anonymous replica could be registered and could prevent WAL files removal (gh-5287).
- XlogGapError is not a critical error anymore. It means, box.info.replication will show upstream status as 'loading' if the error was found. The upstream will be restarted until the error is resolved automatically with a help of another instance, or until the replica is removed from box.cfg.replication (gh-5287).

### LuaJIT

- Fixed the error occurring on loading luajit-gdb.py with Python 2 (gh-4828).

### Lua

- Fixed a bug related to ignoring internal getaddrinfo errors. Now they can be thrown out by Lua socket functions (gh-4138).
- Fixed: import of table.clear() method (gh-5210). Affected versions: all 2.6.* until 2.6.0-53-g09aa813 (exclusive).
- Fixed unhandled Lua error that may lead to memory leaks and inconsistencies in `<space_object>:frommap()`, `<key_def_object>:compare()`, `<merge_source>:select()` (gh-5382).

### Memtx

- Transactional manager for memtx engine (gh-4897).

  Read more:

  – *Transaction mode: MVCC*

  – *use_mvcc_engine configuration parameter*

### Module API

- Get rid of typedef redefinitions for compatibility with C99 (gh-5313).

## Tarantool 2.5.3

Release: v. 2.5.3 Date: 2020-12-30 Tag: 2.5.3-0-gf93e480

### Общие сведения

2.5.3 is the second *stable* version of the 2.5 release series. It introduces one improvement and resolves 19 bugs since 2.5.2.

The "stable" label means that we have all planned features implemented and we see no high-impact issues. However, if you encounter an issue, feel free to report it on GitHub.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Изменения или добавления функциональности

### Репликация

- It is now possible to specify synchro quorum as a function of a number `N` of registered replicas instead of a `const` number, for example:

  ```
  box.cfg{replication_synchro_quorum = "N/2 + 1"}
  ```

  Only the non-anonymous bootstrapped replicas amount to `N`. The expression should respect synchro guarantees: at least 50% of the cluster size + 1. The expression value is re-evaluated automatically inside of Tarantool when new replicas appear or old ones are removed (gh-5446).

### Lua

- Show JSON tokens themselves instead of token names `T_*` in the JSON decoder error messages (gh-4339).
- Show a decoding context in the JSON decoder error messages (gh-4339).

### Build

- Deploy packages for Fedora 32 (gh-4966).
- Deploy packages for Debian Bullseye (gh-5638).

### Исправленные ошибки

### Core

- `fiber.cond:wait()` now correctly throws an error when a fiber is cancelled, instead of ignoring the timeout and returning without any signs of an error (gh-5013).
- Fixed a memory corruption issue, which was most visible on macOS, but could affect any system (gh-5312).
- A dynamic module now gets correctly unloaded from memory in case of an attempt to load a non-existing function from it (gh-5475).
- A `swim:quit()` call now can't result in a crash (gh-4570).
- Snapshot recovery with no JSONPath or multikey indices involved now has normal performance (gh-4774).

### Репликация

- A false-positive "too long WAL write" message no longer appears for synchronous transactions (gh-5139).
- A `box.ctl.wait_rw()` call could return when the instance was not in fact writable due to having foreign synchronous transactions. As a result, there was no proper way to wait until the automatically elected leader would become writable. Now `box.ctl.wait_rw()` works correctly (gh-5440).

### SQL

- Data changes in read-only mode are now forbidden (gh-5231).
- Query execution now does not occasionally raise an unrelated error "Space '0' does not exist" (gh-5592).
- Coinciding names of temporary files (used to store data during execution) having two instances running on the same machine no longer cause a segfault (gh-5537).
- The return value of `ifnull()` built-in function is now of a correct type.
- SQL calling Lua functions with box calls inside can no longer result in a memory corruption (gh-5427).

#### LuaJIT

- Dispatching `__call` metamethod no longer causes address clashing (gh-4518, gh-4649).
- Fixed a false positive panic when yielding in debug hook (gh-5649).

#### Lua

- An attempt to use a `net.box` connection which is not established yet now results in a correctly reported error (gh-4787).
- Fixed a `NULL` dereference on error paths in merger which usually happened on a 'wrong' `key_def` (gh-5450).
- Calling `key_def.compare_with_key()` with an invalid key no longer causes a segfault (gh-5307).
- Fixed a hang which occured when `tarantool` ran a user script with the `-e` option and this script exited with an error (like with `tarantool -e 'assert(false)'`) (gh-4983).

#### Memtx

- The `on_schema_init` triggers now can't cause duplicates in primary key (gh-5304). As part of the fix there is a new function, *box.ctl.is_recovery_finished()*.

### Tarantool 2.5.2

Release: v. 2.5.2 Date: 2020-10-22 Tag: 2.5.2-1-gf63c43b

#### Общие сведения

This release resolves roughly 25 issues since the 2.5.2 version. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

#### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

#### Изменения или добавления функциональности

#### Core

- New function `space:alter(options)` to change some space settings without recreation nor touching `_space` space. *Read more.*

### Module API

- Exposed the box region, key_def and several other functions in order to implement external tuple.keydef and tuple.merger modules on top of them (gh-5273, gh-5384).

### Исправленные ошибки

### Core

- Fixed a bug related to ignoring internal getaddrinfo errors on macOS in logger (gh-4138).
- Fixed a crash when JSON tuple field access was used to get a multikey indexed field, and when a JSON contained [*] in the beginning (gh-5224).
- Fixed msgpack extension types decoding error message (gh-5016).
- Dropped restrictions on nullable multikey index root. They were introduced due to inaccuracy in multikey index realization. It is now fixed. Also all fields are now nullable by default as it was before 2.2.1 (gh-5192).
- Fixed fibers switch-over to prevent JIT machinery misbehavior. Trace recording is aborted when fiber yields the execution. The yield occurring while the compiled code is being run (it's likely a function with a yield underneath called via LuaJIT FFI) leads to the platform panic (gh-1700, gh-4491).
- Fixed fibers switch-over to prevent implicit GC disabling. The yield occurring while user-defined __gc metamethod is running leads to the platform panic.

### Репликация

- Fixed a bug when a rolled back synchronous transaction could become committed after restart (gh-5140).
- Fixed crash in synchronous replication when master's local WAL write fails (gh-5146).
- Instance will terminate if a synchronous transaction confirmation or rollback fail. Before it was undefined behavior (gh-5159).
- Snapshot could contain changes from a rolled back synchronous transaction (gh-5167).
- Fixed a crash when synchronous transaction's rollback and confirm could be written simultaneously for the same LSN (gh-5185).
- Fixed a crash when replica cleared synchronous transaction queue, while it was not empty on master (gh-5195).
- During recovery of synchronous changes from snapshot the instance could crash (gh-5288).
- Having synchronous rows in the snapshot could make the instance hang on recovery (gh-5298).
- Anonymous replica could be registered and could prevent WAL files removal (gh-5287).
- XlogGapError is not a critical error anymore. It means, box.info.replication will show upstream status as 'loading' if the error was found. The upstream will be restarted until the error is resolved automatically with a help of another instance, or until the replica is removed from box.cfg.replication (gh-5287).

### LuaJIT

- Fixed the error occurring on loading luajit-gdb.py with Python 2 (gh-4828).

### Lua

- Fixed a bug related to ignoring internal getaddrinfo errors. Now they can be thrown out by Lua socket functions (gh-4138).
- Fixed: import of table.clear() method (gh-5210). Affected versions: 2.5.0-265-g3af79e70b (inclusive) to 2.5.1-52-ged9a156 (exclusive).
- Fixed unhandled Lua error that may lead to memory leaks and inconsistencies in `<space_object>:frommap()`, `<key_def_object>:compare()`, `<merge_source>:select()` (gh-5382).

### SQL

- SQL view are not alterable anymore. Beforehand it led to undefined behavior.

### Разное

- Fixed potential lag on boot up procedure when system's password database is slow in access (gh-5034).

### Module API

- Get rid of typedef redefinitions for compatibility with C99 (gh-5313).

## Tarantool 2.5.1

Release: v. 2.5.1 Date: 2020-07-17 Tag: 2.5.1-1-g635f6e5

### Общие сведения

2.5.1 is the beta version of the 2.5 release series.

This release introduces roughly 11 features and resolves 34 bugs since the 2.4.1 version. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

Notable changes are:

- Synchronous replication (beta).
- Allow an anonymous replica follow another anonymous replica.
- Fixed numerous crashes in Vinyl.
- Make implicit cast rules for assignment operation more strict in SQL.
- Updated curl version to 7.71.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

There are changes labeled with **[Breaking change]**. It means that the old behavior was considered error-prone and therefore changed to protect users from unintended mistakes. However, there is a little probability that someone can lean on the old behavior, and this label is to bring attention to the things that have been changed.

### Изменения или добавления функциональности

#### Core

- **[Breaking change]** *box.session.push()* parameter `sync` is deprecated and deleted. It does not work anymore, and a usage attempt leads to an error (gh-4689).
- **[Breaking change]** Define new rules for implicit cast for assignment operation in SQL (gh-3809). Read more: *Implicit string/numeric cast*.
- Symbols of the Tarantool executable are not masked anymore. Some private symbols may become visible and available for FFI and `dlopen()` + `dlsym()` (gh-2971).
- Add ability to set up logging early without configuring the `box` engine (gh-689).
- To retrieve information about memory usage, *box.info.memory()* can be used (gh-4688).

#### Vinyl

- `box.snapshot()` now ignores throttling of the scheduler and forces the dump process immediately (gh-3519).

#### SQL

- Use naming pattern "COLUMN_N" for automatically generated column's names (gh-3962). Read more: *Select list*.

#### Репликация

- Add *box.info.replication_anon()*. When called, it lists anonymous replicas in the same format as `box.info.replication`, the only exception is that anonymous replicas are indexed by their uuid strings (gh-4900).
- Allow anonymous replicas to be followed by other ones (gh-4696).
- *Synchronous replication* can be enabled per-space using the `is_sync` space option (gh-4842).

### Build

- Add initial support for OpenBSD (gh-4967).

### Разное

- Don't start the *'example' instance* after installing Tarantool (gh-4507).

  Before this release, the Tarantool package for Debian and Ubuntu automatically enable and start the 'example' instance, which listens on the TCP port 3301. Starting from this release, the instance file is installed to `/etc/tarantool/instances.available/example.lua`, but is not enabled by default and not started anymore. One may perform the following actions to enable and start it:

  ```
  ln -s /etc/tarantool/instances.available/example.lua \
      /etc/tarantool/instances.enabled/example.lua
  systemctl start tarantool@example
  ```

  The existing configuration will not be updated automatically at package update, so manual actions are required to stop and disable the instance (if it is not needed, of course):

  ```
  systemctl stop tarantool@example
  rm /etc/tarantool/instances.enabled/example.lua
  ```

- When LTO is enabled, Luajit is built with it (gh-3743).

### Исправленные ошибки

### Core

- Fixed assert outdated due to multikey index arrival (gh-5132).
- Fixed a bug in altering a normal index to a functional one (n/a).
- Fixed a couple of internal symbols dangling in global namespace _G (gh-4812).
- Fixed bug when on_shutdown triggers were not executed after EOF (gh-4703).
- Fixed inability to handle ULL constants in Lua mode console (gh-4682).
- Fixed a bug in C module reloading (gh-4945).
- Fixed confusing implicit requirements for tuple fields (gh-5027).
- Added needed key validation to space_before_replace (gh-5017).
- Fixed check of index field map size which led to crash (gh-5084).
- Fixed NULL pointer dereference when merger is called via the binary protocol (say, via net.box) (gh-4954).
- Fix wrong mpsgpack extension type in an error message at decoding (gh-5017).
- Fixed crash when invalid JSON was used in update() (gh-5135).

### Репликация

- Fixed possible ER_TUPLE_FOUND error when bootstrapping replicas in an 1.10/2.1.1 cluster (gh-4924).
- Fixed tx boundary check for half-applied txns (gh-5125).
- Fixed replication tx boundaries after local space rework (gh-4928).

### Lua

- Added format string usage to form a CustomError message (gh-4903). Read more: *Custom error*.
- Fixed error while closing socket.tcp_server socket (gh-4087).
- Extended box.error objects reference counter to 64 bit to prevent possible overflow (gh-4902).
- Refactored Lua table encoding: removed excess Lua function object and left protected Lua frame only for the case __serialize is a function to improve `msgpack.encode()` performance (no GH issue).
- Improved Lua call procedure for the case of built-in functions. Prepared GCfunc object is used instead of temporary one, resulting in 3-6% garbage collection reduction.
- Enabled luacheck in continuous integration (no GH issue).
- Fixed warnings (two of them were real bugs!) found by luacheck in a source code (no GH issue).

### SQL

- Fixed wrong order of rows as a result of query containing column of SCALAR type in ORDER BY clause (gh-4697).
- Fixed bug with the display of collation for scalar fields in <SELECT> result, when `sql_full_metadata` is enabled (gh-4755).
- Block using HASH indexes in SQL since scheduler is unable to use it properly (gh-4659).
- Fixed races and corner cases in box (re)configuration (gh-4231).

### Vinyl

- Fixed crash during compaction due to tuples with size exceeding vinyl_max_tuple_size setting (gh-4864).
- Fixed crash during recovery of vinyl index due to the lack of file descriptors (gh-4805).
- Fixed crash during executing upsert changing primary key in debug mode (gh-5005).
- Fixed crash due to triggered dump process during secondary index creation (gh-5042).
- Fixed crash/deadlock (depending on build type) during dump process scheduling and concurrent DDL operation (gh-4821).
- Fixed crash during read of prepared but still not yet not committed statement (gh-3395).
- Fixed squashing set and arithmetic upsert operations (gh-5106).
- Created missing folders for vinyl spaces and indexes if needed to avoid confusing fails of tarantool started from backup (gh-5090).

- Fixed crash during squash of many (more than 4000) upserts modifying the same key (gh-4957).

#### Memtx

- Fixed concurrent replaces on index building. Tuples are now referenced on all needed execution paths (gh-4973).

#### Разное

- Fixed a possible stacked diagnostics crash due to incorrect reference count (gh-4887).

### Tarantool 2.4.3

Release: v. 2.4.3 Date: 2020-10-22 Tag: 2.4.3-1-g986fab7

#### Общие сведения

This release resolves roughly 13 issues since the 2.4.2 version. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

#### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in the binary data layout, client-server protocol, and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

#### Изменения или добавления функциональности

#### Module API

- Exposed the box region, key_def and several other functions in order to implement external tuple.keydef and tuple.merger modules on top of them (gh-5273, gh-5384).

#### Исправленные ошибки

#### Core

- Fixed a crash when JSON tuple field access was used to get a multikey indexed field, and when a JSON contained [*] in the beginning (gh-5224).
- Fixed msgpack extension types decoding error message (gh-5016).
- Dropped restrictions on nullable multikey index root. They were introduced due to inaccuracy in multikey index realization. It is now fixed. Also all fields are now nullable by default as it was before 2.2.1 (gh-5192).

- Fixed fibers switch-over to prevent JIT machinery misbehavior. Trace recording is aborted when fiber yields the execution. The yield occurring while the compiled code is being run (it's likely a function with a yield underneath called via LuaJIT FFI) leads to the platform panic (gh-1700, gh-4491).
- Fixed fibers switch-over to prevent implicit GC disabling. The yield occurring while user-defined __gc metamethod is running leads to the platform panic.

#### Репликация

- Anonymous replica could be registered and could prevent WAL files removal (gh-5287).
- XlogGapError is not a critical error anymore. It means, box.info.replication will show upstream status as 'loading' if the error was found. The upstream will be restarted until the error is resolved automatically with a help of another instance, or until the replica is removed from box.cfg.replication (gh-5287).

#### LuaJIT

- Fixed the error occurring on loading luajit-gdb.py with Python2 (gh-4828).

#### Lua

- Fixed unhandled Lua error that may lead to memory leaks and inconsistencies in `<space_object>:frommap()`, `<key_def_object>:compare()`, `<merge_source>:select()` (gh-5382).

#### SQL

- SQL view are not alterable anymore. Beforehand it led to undefined behavior.

#### Разное

- Fixed potential lag on boot up procedure when system's password database is slow in access (gh-5034).

#### Module API

- Get rid of typedef redefinitions for compatibility with C99 (gh-5313).

### Tarantool 2.4.2

Release: v. 2.4.2 Date: 2020-07-17 Tag: 2.4.2-1-g3f00d29

### Общие сведения

2.4.2 is the first stable version of the 2.4 release series. The label stable means we have all planned features implemented and we see no high-impact issues.

This release resolves roughly 32 issues since the latest beta version. There may be bugs in less common areas, please feel free to file an issue at GitHub.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the box.schema.upgrade() procedure to unlock all the new features of the 2.x series.

### Изменения или добавления функциональности

#### Core

- *box.session.push()* parameter `sync` is deprecated. A warning is printed when the sync is used, but it still works. It is removed in the next version (gh-4689).

### Разное

- Don't start *'example' instance* after installing Tarantool (gh-4507).

  Before this release tarantool package for Debian and Ubuntu automatically enable and start 'example' instance, which listens on the TCP port 3301. Starting from this release the instance file is installed to /etc/tarantool/instances.available/example.lua, but is not enabled by default and not started anymore. One may perform the following actions to enable and start it:

  ```
  ln -s /etc/tarantool/instances.available/example.lua \
      /etc/tarantool/instances.enabled/example.lua
  systemctl start tarantool@example
  ```

  Existing configuration will not be updated automatically at package update, so manual actions are required to stop and disable the instance (if it is not needed, of course):

  ```
  systemctl stop tarantool@example
  rm /etc/tarantool/instances.enabled/example.lua
  ```

### Исправленные ошибки

#### Core

- Fixed a bug in altering a normal index to a functional one (n/a).
- Fixed a couple of internal symbols dangling in global namespace _G (gh-4812).
- Fixed bug when on_shutdown triggers were not executed after EOF (gh-4703).
- Fixed inability to handle ULL constants in Lua mode console (gh-4682).

- Fixed a bug in C module reloading (gh-4945).
- Fixed assert outdated due to multikey index arrival (gh-5132).
- Fixed confusing implicit requirements for tuple fields (gh-5027).
- Added needed key validation to space_before_replace (gh-5017).
- Fixed check of index field map size which led to crash (gh-5084).
- Fixed NULL pointer dereference when merger is called via the binary protocol (say, via net.box) (gh-4954).
- Fix wrong mpsgpack extension type in an error message at decoding (gh-5017).
- Fixed crash when invalid JSON was used in update() (gh-5135).

#### Репликация

- Fixed possible ER_TUPLE_FOUND error when bootstrapping replicas in an 1.10/2.1.1 cluster (gh-4924).
- Fixed tx boundary check for half-applied txns (gh-5125).
- Fixed replication tx boundaries after local space rework (gh-4928).

#### Lua

- Added format string usage to form a CustomError message (gh-4903). Read more: *Custom error*.
- Fixed error while closing socket.tcp_server socket (gh-4087).
- Extended box.error objects reference counter to 64 bit to prevent possible overflow (gh-4902).

#### SQL

- Fix wrong order of rows as a result of query containing column of SCALAR type in ORDER BY clause (gh-4697).
- Fix bug with the display of collation for scalar fields in <SELECT> result, when `sql_full_metadata` is enabled (gh-4755).
- Block using HASH indexes in SQL since scheduler is unable to use it properly (gh-4659).
- Fixed races and corner cases in box (re)configuration (gh-4231).

#### Vinyl

- Fixed crash during compaction due to tuples with size exceeding vinyl_max_tuple_size setting (gh-4864).
- Fixed crash during recovery of vinyl index due to the lack of file descriptors (gh-4805).
- Fixed crash during executing upsert changing primary key in debug mode (gh-5005).
- Fixed crash due to triggered dump process during secondary index creation (gh-5042).
- Fixed crash/deadlock (depending on build type) during dump process scheduling and concurrent DDL operation (gh-4821).

- Fixed crash during read of prepared but not committed statement (gh-3395).
- Fixed squashing set and arithmetic upsert operations (gh-5106).
- Create missing folders for vinyl spaces and indexes if needed to avoid confusing fails of tarantool started from backup (gh-5090).
- Fixed crash during squash of many (more than 4000) upserts modifying the same key (gh-4957).

#### Memtx

- Fixed concurrent replaces on index building. Tuples are now referenced on all needed execution paths (gh-4973).

### Tarantool 2.4.1

Release: v. 2.4.1 Date: 2020-04-20 Tag: 2.4.1-1-g6c75f80

#### Общие сведения

2.4.1 is the beta version of the 2.4 release series.

This release introduces roughly 20 features and resolves 92 bugs since the 2.3.1 version. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

Notable changes are:

- UUID type was introduced
- It is now possible to report stack of errors
- Added popen built-in module
- Create errors of custom type and transparent marshaling over net.box

#### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the box.schema.upgrade() procedure to unlock all the new features of the 2.x series.

#### Изменения или добавления функциональности

#### Core

- Possibility to create errors of a custom user type (gh-4398). Read more:
  - *Error responses*
  - *MessagePack extensions (ERROR type)*
  - *box.error.new()*.
- Transparent marshalling error through net.box (gh-4398). Read more:

  - *box.error.new()*
  - *MessagePack extensions (ERROR type)*.

- `box_session_push()`: a new public C API function. It takes `const char * MessagePack` and returns it to the client out of order, just like the Lua `box.session.push()` does (gh-4734). Read more: *box_session_push()*.
- Introduce a new field type and a new index type: UUID (gh-4268, gh-2916). Read more:
  - *UUID field type*
  - *mapping UUID field to index types*
  - *uuid.new()*
- Add ability to encode/decode UUIDs to MsgPack (gh-4268, gh-2916). Read more: *MessagePack extensions (UUID type)*.
- Introduce stacked diagnostic area: now each Lua table representing an error object features the `.prev` member and the `:set_prev()` method so that errors can be organized into lists. IProto protocol is extended with new command keys to support this feature as well (gh-1148). For details, refer to *error_object*.

### Lua

- Added `popen` built-in module (gh-4031).

  The module provides popen implementation that is integrated with tarantool's event loop (like built-in `fio` and `socket` modules).

  It support bidirectional communication with a process: the module can feed input to a process and capture its output. This way it allows to run streaming programs (like `grep`) and even work interactively with outside REPL (say, `python -i`).

  A key feature of the implementation is that it uses vfork() under hood and so does not copy virtual memory tables. Copying of them may be quite time consuming: os.execute() takes ~2.5 seconds when 80 GiB is allocated for memtx. Moreover, when memory overcommit is disabled (which is default) it would be not possible to fork a process when more then half of available physical memory is mapped to tarantool's process.

  The API should be considered as beta: it is quite basic and will be extended with convenience features. On the other hand, it may be changed in a backward-incompatible manner in the future releases if it will be valuable enough.

  For more details, refer to the `popen` module *documentation*.
- `fio` descriptors are closed on garbage collection (gh-4727). Read more in description of *fio.open()*.
- *fio.tempdir()* uses the `$TMPDIR` environment variable as a path indication to create temporary directories (gh-4794).
- Expose `lua_checktuple` function (gh-2553).
- *fiber.name* maximal length is extended to 255 (gh-4394).
- Add `tarantoolctl rocks` commands: `build`, `config`, `download`, `init`, `lint`, `new_version`, `purge`, `which`, `write_rockspec` (gh-4629). Read more in *Команды для управления модулями Tarantool*.

### Разное

- `box.info.listen`: a new variable in the `box.info`. Shows the real port when bound to the port `0`. For example, if the `listen` parameter of `box.cfg` is set to `127.0.0.1:0`, the `box.info.listen` shows `127.0.0.1:<real_port>` (gh-4620). Read more: *box.info.listen*.
- `sequence:current()`: a new function to get the current sequence value without changing it (gh-4752). Read more: *sequence_object:current()*.

### Исправленные ошибки

### Core

- *fiber.storage* is cleaned between requests, and can be used as a request-local storage. Previously `fiber.storage` could contain some old values in the beginning of an iproto request execution, and it needed to be nullified manually. Now the cleanup is unneeded (gh-4662).
- `tuple`/`space`/`index`:`update()`/`upsert()` were fixed not to turn a value into an infinity when a float value was added to or subtracted from a float column and exceeded the float value range (gh-4701).
- Fix potential execution abort when operating the system runs under heavy memory load (gh-4722).
- Make RTREE indexes handle the out of memory error: before this fix, OOM during the recovery of an RTREE index resulted in segmentation fault (gh-4619).
- Fix the error message returned on using an already dropped sequence (gh-4753).
- Add cancellation guard to avoid WAL thread stuck (gh-4127).
- Fix execution abort when `memtx_memory` and `vinyl_memory` are set to more than 4398046510080 bytes. Now an error message is returned (gh-4705).
- `box.error.new()` does not add a created error to the Tarantool's diagnostic area anymore (gh-4778). Read more:
  - *box.error.new()*
  - *box.error.set()*.
- Add Lua output format support for `box.session.push()` (gh-4686).

### Репликация

- Fix rebootstrap procedure not working in case replica itself is listed in `box.cfg.replication` (gh-4759).
- Fix possible user password leaking via replication logs (gh-4493).
- Refactor `vclock` map to be exactly 4 bytes in size to fit all 32 replicas regardless of the compiler used (see in this commit).
- Fix crash when the replication applier rollbacks a transaction (gh-4730, gh-4776).
- Fix segmentation fault on master side when one of the replicas transitions from anonymous to normal (gh-4731).
- Local space operations are now counted in 0th vclock component. Every instance may have its own 0-th vclock component not matching others'. Local space operations are not replicated at all, even as NOPs (gh-4114).

- Gc consumers are now ordered by their vclocks and not by vclock signatures. Only the WALS that contain no entries needed by any of the consumers are deleted (gh-4114).

### Lua

- json: `:decode()` does not spoil instance's options with per-call ones (when it is called with the second argument) (gh-4761).
- Handle empty input for `uri.format()` properly (gh-4779).
- `os.environ()` is now changed when `os.setenv()` is called (gh-4733).
- `netbox.self:call/eval()` now returns the same types as `netbox_connection:call/eval`. Previously it could return a tuple or `box.error` cdata (gh-4513).
- `box.tuple.*` namespace is cleaned up from private functions. *box.tuple.is()* description is added to documentation (gh-4684).
- `tarantoolctl rocks search`: fix the `--all` flag (gh-4529).
- `tarantoolctl rocks remove`: fix the `--force` flag (gh-3632).
- libev: backport fix for listening for more then 1024 file descriptors on Mac OS (gh-3867).

### SQL

- Fix `box.stat()` behavior: now it collects statistics on the `PREPARE` and `EXECUTE` methods as expected (gh-4756).
- Add ability to drop any table constraint using the following statement: `ALTER TABLE <table_name> DROP CONSTRAINT <constraint_name>`. Previously, it was possible to drop only foreign key constraints with such a statement (gh-4120). Read more in *Alter Table*.
- "No such constraint" error now contains the name of the table this constraint belongs to.
- Add an empty body to the `UNPREPARE` IProto response (gh-4769).
- Reset all the placeholders' bound values after execution of a prepared statement (gh-4825).
- The inserted values are inserted in the order in which they are given in case of INSERT into space with autoincrement (gh-4256).

### Types related changes

- Rework the `NUMBER` type in SQL: now it completely matches with NoSQL definition. Integers inserted into a field of the `NUMBER` type are no longer forced to floating point representation; `CAST AS NUMBER` operation applied to an integer value doesn't change it; `CAST AS NUMBER` operation applied to a boolean value now converts it into `1`, `0` or `NULL` depending on particular boolean value (gh-4233). Read more in *Data Type Conversion*.
- Fix the `CAST AS NUMBER` operation applied to blob values: floating point representation of the result is no longer forced (gh-4463).
- Fix integer overflow error during addition of the integer and floating point values stored in the `SCALAR` column (gh-4369).
- Explicit and implicit cast from string containing floating point value to integer or unsigned types are disallowed.

- Maximum length of a blob value that is allowed to be cast to integer or unsigned types are limited to 12287 bytes.
- Fix wrong result of CAST() operator from blob to integer type in case a blob value does not have terminating '\0' (gh-4766).

### HTTP client

- When building Tarantool with bundled `libcurl`, link it with the `c-ares` library by default (gh-4591).

### LuaJIT

- `__pairs`/`__ipairs` metamethods handling is removed since we faced the issues with the backward compatibility between Lua 5.1 and Lua 5.2 within Tarantool modules as well as other third party code (gh-4770).
- Introduce `luajit-gdb.py` extension with commands for inspecting LuaJIT internals. The extension obliges one to provide gdbinfo for `libluajit`, otherwise loading fails. The extension provides the following commands:
  - `lj-arch` dumps values of LJ_64 and LJ_GC64 macro definitions
  - `lj-tv` dumps the type and GCobj info related to the given TValue
  - `lj-str` dumps the contents of the given GCstr
  - `lj-tab` dumps the contents of the given GCtab
  - `lj-stack` dumps Lua stack of the given lua_State
  - `lj-state` shows current VM, GC and JIT states
  - `lj-gc` shows current GC stats
- Fix string to number conversion: current implementation respects the buffer length (gh-4773).
- "FFI sandwich" (*) detection is introduced. If sandwich is detected while trace recording the recording is aborted. The sandwich detected while mcode execution leads to the platform panic.
- luaJIT_setmode call is prohibited while mcode execution and leads to the platform panic.

(*) The following stack mix is called FFI sandwich: `Lua-FFI -> C routine -> Lua-C API -> Lua VM`.

This sort of re-entrancy is explicitly not supported by LuaJIT compiler. For more info see gh-4427.

### Vinyl

- Fix assertion fault due to triggered dump process during secondary index build (gh-4810).

#### Разное

- Fix crashes at attempts to use `-e` and `-l` command line options concatenated with their values, like this: `-eprint(100)` (gh-4775).
- Fix inability to upgrade from 2.1 if there was an automatically generated sequence (gh-4771).
- Prettify the error message for `user.grant()`: no extra ' ' for universal privileges (gh-714).
- Update `libopenssl` version to 1.1.1f since the previous one was EOLed (gh-4830).

#### Building from sources

- Update the `decNumber` library to silence the build warning produced on too long integer constant (see in this commit).
- Fix static build (`-DBUILD_STATIC=ON`) when `libunwind` depends on `liblzma` (gh-4551).

### Tarantool 2.3.3

Release: v. 2.3.3 Date: 2020-07-17 Tag: 2.3.3-1-g43af95e

#### Общие сведения

2.3.3 is the last stable version of the 2.3 release series. The label stable means we have all planned features implemented and we see no high-impact issues.

This release resolves roughly 26 issues since the latest stable version. There may be bugs in less common areas, please feel free to file an issue at GitHub.

Please note, this release contains **no new features**.

#### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the box.schema.upgrade() procedure to unlock all the new features of the 2.x series.

#### Исправленные ошибки

#### Core

- Fixed a bug in altering a normal index to a functional one (n/a).
- Fixed a couple of internal symbols dangling in global namespace _G (gh-4812).
- Fixed bug when on_shutdown triggers were not executed after EOF (gh-4703).
- Fixed a bug in C module reloading (gh-4945).
- Fixed assert outdated due to multikey index arrival (gh-5132).
- Fixed confusing implicit requirements for tuple fields (gh-5027).

- Added needed key validation to space_before_replace (gh-5017).
- Fixed check of index field map size which led to crash (gh-5084).
- Fixed NULL pointer dereference when merger is called via the binary protocol (say, via net.box) (gh-4954).
- Fixed crash when invalid JSON was used in update() (gh-5135).

### Репликация

- Fixed possible ER_TUPLE_FOUND error when bootstrapping replicas in an 1.10/2.1.1 cluster (gh-4924).
- Fixed tx boundary check for half-applied txns (gh-5125).
- Fixed replication tx boundaries after local space rework (gh-4928).

### Lua

- Fixed error while closing socket.tcp_server socket (gh-4087).

### SQL

- Fixed wrong order of rows as a result of query containing column of SCALAR type in ORDER BY clause (gh-4697).
- Fixed bug with the display of collation for scalar fields in <SELECT> result, when `sql_full_metadata` is enabled (gh-4755).
- Block using HASH indexes in SQL since scheduler is unable to use it properly (gh-4659).
- Fixed races and corner cases in box (re)configuration (gh-4231).

### Vinyl

- Fixed crash during compaction due to tuples with size exceeding vinyl_max_tuple_size setting (gh-4864).
- Fixed crash during recovery of vinyl index due to the lack of file descriptors (gh-4805).
- Fixed crash during executing upsert changing primary key in debug mode (gh-5005).
- Fixed crash due to triggered dump process during secondary index creation (gh-5042).
- Fixed crash/deadlock (depending on build type) during dump process scheduling and concurrent DDL operation (gh-4821).
- Fixed crash during read of prepared but still not yet not committed statement (gh-3395).
- Fixed squashing set and arithmetic upsert operations (gh-5106).
- Create missing folders for vinyl spaces and indexes if needed to avoid confusing fails of tarantool started from backup (gh-5090).
- Fixed crash during squash of many (more than 4000) upserts modifying the same key (gh-4957).

### Memtx

- Fixed concurrent replaces on index building. Tuples are now referenced on all needed execution paths (gh-4973).

### Tarantool 2.3.2

Release: v. 2.3.2 Date: 2020-04-20 Tag: 2.3.2-1-g9be641b

### Общие сведения

2.3.2 is the first stable version of the 2.3 release series. The label stable means we have all planned features implemented and we see no high-impact issues.

This release resolves roughly 39 issues since the latest beta version. There may be bugs in less common areas, please feel free to file an issue at GitHub.

Please note, this release contains **no new features**.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the box.schema.upgrade() procedure to unlock all the new features of the 2.x series.

### Исправленные ошибки

### Core

- *fiber.storage* is cleaned between requests, and can be used as a request-local storage. Previously `fiber.storage` could contain some old values in the beginning of an iproto request execution, and it needed to be nullified manually. Now the cleanup is unneeded (gh-4662).
- `tuple`/`space`/`index`:`update()`/`upsert()` were fixed not to turn a value into an infinity when a float value was added to or subtracted from a float column and exceeded the float value range (gh-4701).
- Fix potential execution abort when operating the system runs under heavy memory load (gh-4722).
- Make RTREE indexes handle the out of memory error: before this fix, OOM during the recovery of an RTREE index resulted in segmentation fault (gh-4619).
- Fix the error message returned on using an already dropped sequence (gh-4753).
- Add cancellation guard to avoid WAL thread stuck (gh-4127).
- Fix execution abort when `memtx_memory` and `vinyl_memory` are set to more than 4398046510080 bytes. Now an error message is returned (gh-4705).
- Add Lua output format support for `box.session.push()` (gh-4686).

### Репликация

- Fix rebootstrap procedure not working in case replica itself is listed in `box.cfg.replication` (gh-4759).
- Fix possible user password leaking via replication logs (gh-4493).
- Refactor vclock map to be exactly 4 bytes in size to fit all 32 replicas regardless of the compiler used (https://github.com/tarantool/tarantool/commit/e5679980aa5f813553a95ab7d31f111dd0893df6).
- Fix crash when the replication applier rollbacks a transaction (gh-4730, gh-4776).
- Fix segmentation fault on master side when one of the replicas transitions from anonymous to normal (gh-4731).
- Local space operations are now counted in 0th vclock component. Every instance may have its own 0-th vclock component not matching others'. Local space operations are not replicated at all, even as NOPs (gh-4114).
- Gc consumers are now ordered by their vclocks and not by vclock signatures. Only the WALS that contain no entries needed by any of the consumers are deleted (gh-4114).

### Lua

- json: `:decode()` does not spoil instance's options with per-call ones (when it is called with the second argument) (gh-4761).
- Handle empty input for `uri.format()` properly (gh-4779).
- `os.environ()` is now changed when `os.setenv()` is called (gh-4733).
- `netbox.self:call/eval()` now returns the same types as `netbox_connection:call/eval`. Previously it could return a tuple or `box.error` cdata (gh-4513).
- `box.tuple.*` namespace is cleaned up from private functions. *box.tuple.is()* description is added to documentation (gh-4684).
- `tarantoolctl rocks search`: fix the `--all` flag (gh-4529).
- `tarantoolctl rocks remove`: fix the `--force` flag (gh-3632).
- libev: backport fix for listening for more then 1024 file descriptors on Mac OS (gh-3867).

### SQL

- Fix `box.stat()` behavior: now it collects statistics on the `PREPARE` and `EXECUTE` methods as expected (gh-4756).
- Add an empty body to the `UNPREPARE` IProto response (gh-4769).
- Reset all the placeholders' bound values after execution of a prepared statement (gh-4825).
- The inserted values are inserted in the order in which they are given in case of INSERT into space with autoincrement (gh-4256).

### HTTP client

- When building Tarantool with bundled `libcurl`, link it with the `c-ares` library by default (gh-4591).

### LuaJIT

- `__pairs`/`__ipairs` metamethods handling is removed since we faced the issues with the backward compatibility between Lua 5.1 and Lua 5.2 within Tarantool modules as well as other third party code (gh-4770).
- Introduce `luajit-gdb.py` extension with commands for inspecting LuaJIT internals. The extension obliges one to provide gdbinfo for `libluajit`, otherwise loading fails. The extension provides the following commands:
  - `lj-arch` dumps values of LJ_64 and LJ_GC64 macro definitions
  - `lj-tv` dumps the type and GCobj info related to the given TValue
  - `lj-str` dumps the contents of the given GCstr
  - `lj-tab` dumps the contents of the given GCtab
  - `lj-stack` dumps Lua stack of the given lua_State
  - `lj-state` shows current VM, GC and JIT states
  - `lj-gc` shows current GC stats
- Fix string to number conversion: current implementation respects the buffer length (gh-4773).
- "FFI sandwich" (*) detection is introduced. If sandwich is detected while trace recording the recording is aborted. The sandwich detected while mcode execution leads to the platform panic.
- luaJIT_setmode call is prohibited while mcode execution and leads to the platform panic.

(*) The following stack mix is called FFI sandwich:

    Lua-FFI -> C routine -> Lua-C API -> Lua VM

This sort of re-entrancy is explicitly not supported by LuaJIT compiler. For more info see gh-4427.

### Vinyl

- Fix assertion fault due to triggered dump process during secondary index build (gh-4810).

### Разное

- Fix crashes at attempts to use `-e` and `-l` command line options concatenated with their values, like this: `-eprint(100)` (gh-4775).
- Fix inability to upgrade from 2.1 if there was an automatically generated sequence (gh-4771).
- Update `libopenssl` version to 1.1.1f since the previous one was EOLed (gh-4830).

### Building from sources

- Update the `decNumber` library to silence the build warning produced on too long integer constant (https://github.com/tarantool/tarantool/commit/aab03a735c7a215b4371ef834f7d08432b1bf0f7).
- Fix static build (`-DBUILD_STATIC=ON`) when `libunwind` depends on `liblzma` (gh-4551).

## Tarantool 2.3.1

Release: v. 2.3.1 Date: 2019-12-31 Tag: 2.3.1-0-g5a1a220

### Общие сведения

2.3.1 is the beta version of the 2.3 release series.

This release introduces roughly 38 features and resolves 102 bugs since the 2.2.1 version. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

Notable changes are:

- field name and JSON path updates
- anonymous replica
- new DOUBLE SQL type (and new 'double' box field type)
- stored and indexed decimals (and new 'decimal' field type)
- `fiber.top()`
- feed data from a memory during replica initial join
- SQL prepared statements
- sessions settings service space

Aside of that many other features have been implemented and considerable amount of bugs have been fixed.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Изменения или добавления функциональности

There are changes labeled with **[Breaking change]**. It means that the old behaviour was considered error-prone and therefore changed to protect users from unintended mistakes. However, there is a little probability that someone can lean on the old behaviour, and this label is to bring attention to the things that have been changed.

### SQL

- Introduce prepared statements support and prepared statements cache (gh-2592, gh-3292). Using of prepared statements allows to eliminate overhead of transmitting a statement text over a network and parsing it each time before execution. Aside of this, it allows to acquire binding parameters and result set columns metainformation prior to actual execution of a statement. This feature is vital for implementing standard DBMS APIs, such as ODBC and JDBC.
- **[Breaking change]** Introduce _session_setting service space as replacement for PRAGMA keyword (gh-4511). All frontends (C, Lua, SQL, binary protocol) can use this space to access or update session settings. Removed `count_changes`, `short_column_names`, `sql_compound_select_limit`, `vdbe_addoptrace` pragmas. Transformed others into _session_settings tuples.
- Extend SQL result set metadata (gh-4407), In addition to the `name` and `type` fields, the `collation`, `is_nullable`, `is_autoincrement`, and `span` fields are added. These new fields are shown when the `full_metadata` session setting is enabled but always sent via binary protocol.
- Add an ability to disable check constraints (gh-4244). Example: `ALTER TABLE foo {ENABLE|DISABLE} CHECK CONSTRAINT bar;`. For details of using from Lua, refer to documentation.
- AUTOINCREMENT for multipart primary key (gh-4217). The auto-increment feature can be set to any `INTEGER` or `UNSIGNED` field of `PRIMARY KEY` using one of the two ways:

  1. `AUTOINCREMENT` in column definition:

  ```
  CREATE TABLE t (i INT, a INT AUTOINCREMENT, PRIMARY KEY (i, a));
  CREATE TABLE t (i INT AUTOINCREMENT, a INT, PRIMARY KEY (i, a));
  ```

  2. `AUTOINCREMENT` in `PRIMARY KEY` definition:

  ```
  CREATE TABLE t (i INT, a INT, PRIMARY KEY (i, a AUTOINCREMENT));
  CREATE TABLE t (i INT, a INT, PRIMARY KEY (i AUTOINCREMENT, a));
  ```

- Allow to create a view from any CTE (common table expression) using `WITH` clause (gh-4149).
- Support user-defined functions in SQL. `box.schema.func.create()` API has been extended and should be used to make some function available in SQL. For details on fields added, refer to the description here: (doc-879). Usage of legacy mechanism`box.internal.sql_function_create` is forbidden now (gh-2200, gh-2233, gh-4113).
- Scalar functions `MIN/MAX` are renamed to `LEAST/GREATEST` (gh-4405)
- Introduce `WITH ENGINE` clause for `CREATE TABLE` statement (gh-4422). To allow a user to specify engine as per table option, `CREATE TABLE` statement has been extended with optional `WITH ENGINE = <engine_name>` clause. This clause comes at the end of `CREATE TABLE` statement. For example:

  ```
  CREATE TABLE t_vinyl (id INT PRIMARY KEY) WITH ENGINE = 'vinyl';
  ```

  Refer also to SQL reference documentation.
- Introduce `DOUBLE` type (gh-3812).
- Display line and position in syntax errors (gh-2611).
- Make constraint names unique within a table (gh-3503). The SQL standard requires `PRIMARY KEY`, `UNIQUE`, `FOREIGN KEY` and `CHECK` constraints to have the unique name within a table. Now Tarantool/SQL follows this requirement. Please refer to (doc-1053).

- Optimization: a tuple already stores a map of offsets of indexed values. After the change, when a field after an indexed field is accessed, the tuple is decoded from the indexed field rather then from beginning (gh-4267).

### Core

- **[Breaking change]** Drop `rows_per_wal` option of `box.cfg()` in favor of `wal_max_size` (gh-3762).
- Decimals can now be stored in spaces. The corresponding field type is introduced: `decimal`. Decimal values are also allowed in the `scalar`, `any`, and `number` fields. Decimal values can be indexed (gh-4333). Also refer to documentation on
  - data types
  - <space_object>:create_index()
- Add support for decimals in update operations (gh-4413). `tuple:update()` and `<space_object>:update()` now support decimal operands for arithmetic operations ('+' and '-'). The syntax is as usual, for example:

```
tarantool> d = box.tuple.new(decimal.new('1'))
---
...
tarantool> d:update{{'+', 1, decimal.new('0.5')}}
---
- [1.5]
...
```

  Insertion ('!') and assignment ('=') are also supported. See also the full description of the `update()` function in documentation.
- Allow to encode/decode decimals to `MsgPack` and to encode to `YAML` and `JSON`. Part of (gh-4333); 485439e3; documentation: (doc-992).
- Introduce field name and JSON path updates (gh-1261).

  Example of update by a field name: `box.space.test:update({{'=', 'foo', 42}})`.

  JSON path update allows to change a value that is nested inside an array or a map. It provides convenient syntax (that is also available for connectors), consumes less space in WAL than replace, and is faster than replaces written in Lua. Example: `box.space.test:update({{'=', 'foo.bar[1]', 42}})`. Please refer to documentation here: (doc-1051).
- Introduce `double` field type. Part of (gh-3812). Though is not very usable in Lua, this new field type has been added in `box` as a base for the SQL `DOUBLE` type.
- vinyl: don't pin index for iterator lifetime (prerequisite for snapshot iterators). 02da82ea
- vinyl: don't exempt dropped indexes from dump and compaction (prerequisite for snapshot iterators). d7387ec9

### Репликация

- `box.info().replication` shows applier/replay's latest error message. Now it also shows the errno description for system errors when it's applicable (gh-4402).
- Feed data from a memory during replica initial join (gh-1271). Aside of obvious speed up from reading from a memory instead of a disk, a read view that is acquired to perform an initial join may be a way more recent, that eliminates the need to play all xlog files since a last snapshot. Now relay need to send only changes that occur during initial join to finally join a replica.
- Introduce a new replica type - anonymous replica (gh-3186). Anonymous replica is not present in cluster space and so there is no limitation for its count in a replica set. Anonymous replica is read-only, but can be deanonymized and enabled for writes. Please refer to documentation: (doc-1050) for API and details.

### Lua

- Expose `require('tarantool').package` which is 'Tarantool' for the community version and 'Tarantool Enterprise' for the enterprise version (gh-4408). This value is already displayed in a console greeting and in `box.info().package`, but it was not accessible from Lua before the first `box.cfg{<...>}` call.
- decimal: add modulo operator (`decimal.new(172.51) % 1 == 0.51`), part of (gh-4403).
- **[Breaking change]** JSON and msgpack serializers now raise an error when a depth of data nesting exceeds the `encode_max_depth` option value. The default value of the option has been increased from 32 to 128. The `encode_deep_as_nil` option is added to give an ability to set the old behaviour back (gh-4434). Notes:
  - These options can be set by using `json.cfg({<...>})` or `msgpack.cfg({<...>})`.
  - `box` data modification functions (`insert`, `replace`, `update` and `upsert`) follow the options of the default msgpack serializer instance, and now these functions raise an error on too many levels of nested data by default rather than cut the data silently. This behaviour can be configured using `msgpack.cfg({<...>})`.
  - previously,`box.tuple.new()`, `space:update()`, `space:upsert()` and several other functions did not follow `encode_max_depth` option; now they do (see also the **Bug fixed** section).
  - previously,`json.cfg` and `msgpack.cfg` tables was not updated when an option had changed; now they show actual values (see also the **Bug fixed** section).
- Show line and column in `json.decode()` errors (gh-3316).
- Exit gracefully when a main script throws an error: notify systemd, log the error (gh-4382).
- key_def: accept both `field` and `fieldno` in `key_def.new(<...>)` (gh-4519). Originally `key_def.new(<...>)` accepted only `fieldno` to allow creation with `<index_object>.parts` as argument. However, index definition format (`<space_object>.create_index(<...>)`) is different and requires `field`. Now both are supported.
- Enable `__pairs` and `__ipairs` metamethods from Lua 5.2 (gh-4560). We still conform Lua 5.1 API that is not always compatible with Lua 5.2. The change is only about those metamethods.
- Implement a new function `fiber.top()`. It returns a table with all fibers alive and lists their CPU consumption. For details, refer to documentation. (gh-2694)
- Expose `errno` field for box.error objects representing system errors. Part of (gh-4402).

### HTTP client

- Add `accept_encoding` option for HTTP client. For details, refer to description here: (doc-1036). (gh-4232).
- Add proxy server related options for HTTP client:
  - `proxy`
  - `proxy_port`
  - `proxy_user_pwd`
  - `no_proxy` For details, refer to description here: doc-896 (gh-4477, gh-4472).

### Разное

- tarantoolctl: allow to start instances with delayed `box.cfg{}` (gh-4435).
- Add package builds and deployment for the following Linux distros:
  - Ubuntu 19.10 Eoan Ermine (gh-4583)
  - CentOS 8 (gh-4543)
  - Fedora 31 (gh-4612)

### Исправленные ошибки

### SQL

- Modify type of a binding value in query response metainformation: always return INTEGER rather than UNSIGNED, even for positive values. This is necessary for consistency with integer literal types. b7d595ac.
- Reuse noSQL way to compare floating point values with integral ones. This allows to handle corner cases like `SELECT 18446744073709551615.0 > 18446744073709551615` uniformly. 73a4a525.
- Create or alter a table with a foreign key may lead to wrong bytecode generation that may cause a crash or wrong result (gh-4495).
- Allow to update a scalar value using SQL in a space that was created from Lua and contains `array`, `map` or `any` fields (gh-4189). Note: Tarantool/SQL provides operations on scalar types and does not support 'array' and 'map' per se.
- Allow nil to be returned from user-defined function (created with `box.schema.func.create()`). 1b39cbcf
- Don't drop a manually created sequence in DROP TABLE statement. a1155c8b
- Remove grants associated with the table in DROP TABLE statement (gh-4546).
- Fix segfault in `sql_expr_coll()` when `SUBSTR()` is called without arguments. 4c13972f
- Fix converting of floating point values from range [2^63, 2^64] to integer (gh-4526).
- Make type string case lower everywhere: in error messages, meta headers, and results of the `typeof()` SQL function. ee60d31d
- Make the`LENGTH()` function to accept boolean argument (gh-4462).

- Make implicit cast from BOOLEAN to TEXT to return uppercase for consistency with explicit cast (gh-4462).
- Fix segfault on binding a value as LIKE argument (gh-4566).
- For user-defined functions, verify that the returned value is of the type specified in the function definition (gh-4387).
- Start using comprehensive serializer `luaL_tofield()` to prepare LUA arguments for user-defined functions. This allows to support cdata types returned from Lua function (gh-4387).
- An error is raised when a user-defined function returns too many values (gh-4387).
- Store a name of user-defined function in VDBE program instead of pointer. This allows to normally handle the situation when a user-defined function has been deleted to the moment of the VDBE code execution (gh-4176).
- Fix casting of VARBINARY value to a NUMBER (gh-4356)
- Print the data type instead of the data itself in `diag_set()` in case of binary data. The reason of this patch is that `LibYAML` converts the whole error message to `base64` in case of non-printable symbols. Part of (gh-4356).
- Remove `ENGINE` from the list of the reserved keywords and allow to use it for identifiers: we are going to use the word as a name of some fields for tables forming informational schema.
- Fix segfault when `LEAST()` or `GREATEST()` built-in function is invoked without arguments (gh-4453).
- Fix dirty memory access when constructing query plan involving search of floating point value in index over integer field (gh-4558).
- `INDEXED BY` clause now obligates the query planner to choose provided index. 49fedfe3

### Core

- Make functional index creation transactional (gh-4401)
- Detect a new invalid JSON path case (gh-4419)
- Randomize the next checkpoint time after manual `box.snapshot()` execution also (gh-4432).
- Fix memory leak in call/eval in case of a transaction is not committed (gh-4388)
- Eliminate warning re `strip_core` option of `box.cfg()` on MacOS and FreeBSD (gh-4464)
- The msgpack serializer that is under `box.tuple.new()` (called tuple serializer) now reflects options set by `msgpack.cfg({<...>})`. Part of (gh-4434). Aside of `box.tuple.new()` behaviour itself, it may affect `tuple:frommap()`, methods of key_def Lua module, tuple and table merger sources, net.box results of `:select()` and `:execute()` calls, and xlog Lua module.
- `box` functions `update` and `upsert` now follow `msgpack.cfg({encode_max_depth = <...>}` option. Part of (gh-4434).
- fiber: make sure the guard page is created; refuse to create a new fiber otherwise (gh-4541). It is possible in case of heavy memory usage, say, when there is no resources to split VMAs.
- recovery: build secondary indices in the hot standby mode without waiting till the main instance termination (gh-4135).
- Fix error message for incorrect return value of functional index extractor function (gh-4553).
  - Was: “Key format doesn’t match one defined in functional index ‘’ of space ‘’: supplied key type is invalid: expected boolean”

  - Now: "<...>: expected array"

- JSON path index now consider is_nullable property when a space had a format (gh-4520).
- Forbid `00000000-0000-0000-0000-000000000000` as the value of `box.cfg({<...>})` options: `replicaset_uuid` and `instance_uuid` (gh-4282). It did not work as expected: the nil UUID was treated as absence of the value.
- Update cache of universe privileges without reconnect (gh-2763).
- net.box: fix memory leak in `net_box:connect(<URI>)` (gh-4588).
- net.box: don't fire the `on_connect` trigger on schema update (gh-4593). Also don't fire the `on_disconnect` trigger if a connection never entered into the `active` state (e.g. when the first schema fetch is failed).
- func: fix use-after-free on function unload. fa2893ea
- Don't destroy a session until `box.session.on_disconnect(<...>)` triggers are finished (gh-4627). This means, for example, that `box.session.id()` can be safely invoked from the `on_disconnect` trigger. Before this change `box.session.id()` returned garbage (usually 0) after yield in the `on_disconnect` trigger. *Note*: tarantool/queue module is affected by this problem in some scenarios. It is especially suggested to update Tarantool at least to this release if you're using this module.
- func: Fix `box.schema.func.drop(<..>)` to unload unused modules (gh-4648). Also fix `box.schema.func.create(<..>)` to avoid loading a module again when another function from the module is loaded.
- Encode Lua number -2^63 as integer in `msgpack.encode()` and box's functions (gh-4672).
- Forbid to drop admin's universe access. 2de398ff. Bootstrap and recovery work on behalf of admin and should be able to fill in the system spaces. Drop of admin's access may lead to an unrecoverable cluster.
- Refactor rope library to eliminate virtual calls to increase performance of the library (mainly for JSON path updates). baa4659c
- Refactor update operation code to avoid extra region-related arguments to take some performance boost (mainly for JSON path updates). dba9dba7
- Error logging has been removed in `engine_find()` to get rid of the error message duplication. 35177fe0.
- decimal: Fix encoding of numbers with positive exponent. Follow-up (gh-692).
- Increment schema version on DDL operations where it did not performed before: alter of trigger, check constraint and foreign key constraint. Part of (gh-2592).

### Репликация

- Stop relay on subscribe error (gh-4399).
- Set `last_row_time` to `now` in `relay_new` and `relay_start` (gh-4431).
- Do not abort replication on ER_UNKNOWN_REPLICA (gh-4455).
- Enter orphan mode on manual replication configuration change (gh-4424).
- Disallow bootstrap of read-only masters (gh-4321).
- Prefer to bootstrap a replica from a fully bootstrapped instance rather than from an instance that is in the process of bootstrapping (gh-4527). This change enables the case when two nodes (B, C) are being bootstrapped simultaneously using the one that is already bootstrapped (A), while A is configured to replicate from {B, C} and B – from {A, C}.

- Return immediately from `box.cfg{<...>}` when an instance is reconfigured with `replication_connect_quorum = 0` (gh-3760). This change also fixes the behaviour of reconfiguration with non-zero `replication_connect_quorum`: `box.cfg{<...>}` returns immediately regardless of whether connections to upstreams are established.
- Apply replication settings of `box.cfg({<...>})` in a strict order (gh-4433).
- Auto reconnect a replica if password is invalid (gh-4550).
- `box.session.su(<username>)` now correctly reports an error for `<username>` longer than `BOX_NAME_MAX` which is 65000. 8b6bdb43
  - Was: 'C++ exception'
  - Now: 'name length <...> is greater than BOX_NAME_MAX'
- Use empty password when a URI in `box.cfg{replication = <...>}` is like `login@host:port` (gh-4605). The behaviour matches the net.box's one now. Explicit `login:@host:port` was necessary before, otherwise a replica displayed the following error: > Missing mandatory field 'tuple' in request
- Fix segfault during replication configuration (`box.cfg{replication = <...>}` call) (gh-4440, gh-4576, gh-4586, gh-4643).
- Cancel a replica joining thread forcefully on Tarantool instance exit (gh-4528).
- Fix the applier to run the `<space>.before_replace` trigger during initial join (gh-4417).

### Lua

- Fix segfault on `ffi.C._say()` without filename (gh-4336).
- Fix `pwd.getpwall()` and `pwd.getgrall()` hang on CentOS 6 and FreeBSD 12 (gh-4428, gh-4447).
- json.encode() now follows `encode_max_depth` option for arrays that leads to a segfault on recursive Lua tables with numeric keys (gh-4366).
- fio.mktree() now reports an error for existing non-directory file (gh-4439).
- `json.cfg` and `msgpack.cfg` tables were not updated when an option is changed. Part of (gh-4434).
- Fix handling of a socket read error in the console client (`console.connect(<URI>)` or `tarantoolctl connect/enter <...>`). 89ec1d97
- Handle the "not enough memory" error gracefully when it is raised from `lua_newthread()` (gh-4556). There are several cases when a new Lua thread is created:
  - Start executing a Lua function call or an eval request (from a binary protocol, SQL or with `box.func.<...>:call()`).
  - Create of a new fiber.
  - Start execution of a trigger.
  - Start of encoding into a YAML format (`yaml.encode()`).
- Fix stack-use-after-scope in `json.decode()` (gh-4637).
- Allow to register several functions using `box.schema.func.create()`, whose names are different only in letters case (gh-4561). This make function names work consistently with other names in tarantool (except SQL, of course).
- Fix decimal comparison with nil. Follow-up (gh-692).
- Fix decimal comparison with `box.NULL` (gh-4454).

- A pointer returned by `msgpack.decode*(cdata<[char] const *>)` functions can be assigned to buffer.rpos now (and the same for msgpackffi) (gh-3926). All those functions now return `cdata<char *>` or `cdata<const char *>` depending of a passed argument. Example of the code that did not work: `res, buf.rpos = msgpack.decode(buf.rpos, buf:size())`.
- lua/pickle: fix typo that leads to reject of negative integers for 'i' (integer) and 'N' (big-endian integer) formats in pickle.pack(). e2d9f664

### HTTP client

- Use bundled `libcurl` rather than system-wide by default. (gh-4318, gh-4180, gh-4288, gh-4389, gh-4397). This closes several known problems that were fixed in recent libcurl versions, including segfaults, hangs, memory leaks and performance problems.
- Fix assertion fail after a curl write error (gh-4232).
- Disable verbose mode when `{verbose = false}` is passed. 72613bb0

### Console Lua output

A new Lua output format is still in the alpha stage and has the known flaws, but we are working to make it rich and stable.

- Output `box.NULL` as `"box.NULL"` rather than `"cdata<void *>: NULL"`, part of (gh-3834) (in quotes for now, yes, due to (gh-4585)
- Add semicolon (`;`) as responses delimiter (EOS, end of stream/statement), analogue of YAMLs end-of-document (`...`) marker. This is vital for remote clients to determine the end of a particular response, part of (gh-3834).
- Fix hang in the console client (`console.connect(<URI>)` or `tarantoolctl connect/enter <...>`) after `\set output lua[,block]` command, part of (gh-3834). In order to overcome it, two changes have been made:
  - Parse `\set output lua[,block]` command on a client prior to sending it to a server, store current responses delimiter (EOS) and use it to determine end of responses.
  - Send `\set output <...>` command with a default output mode when establishing a connection (it is matter if different default modes are set).
- Provide an ability to get or set current responses delimiter using `console.eos([<...>])`, part of (gh-3834).

### LuaJIT

- Fix fold machinery misbehaves (gh-4376).
- Fix for `debug.getinfo(1,'>S')` (gh-3833).
- Fix `string.find` recording (gh-4476).
- Fix the "Data segment size exceeds process limit" error on FreeBSD/x64: do not change resource limits when it is not necessary (gh-4537).
- fold: keep type of emitted CONV in sync with its mode. LuaJIT#524 This fixes the following assertion fail: > asm_conv: Assertion `((IRType)((ir->t).irt & IRT_TYPE)) != st' failed

### Разное

- Support `systemd`'s NOTIFY_SOCKET on OS X (gh-4436).
- Fix linking with static `openssl` library (gh-4437).
- Get rid of warning re empty `NOTIFY_SOCKET` variable (gh-4305).
- rocks: fix 'invalid date format' error when installing a packed rock (gh-4481).
- Remove libyaml from rpm/deb dependencies, because we use bunbled version of libyaml for the packages (since 2.2.1) (gh-4442).
- Fix CLI boolean options handling in `tarantoolctl cat <...>`, such as `--show-system` (gh-4076).
- Fix segfault (out of bounds access) when a stack unwinding error occurs at backtrace printing (gh-4636). Backtrace is printed on the SIGFPE and SIGSEGV signals or when LuaJIT finds itself in the unrecoverable state (`lua_atpanic()`).
- Clear terminal state on panic (gh-4466).
- access: fix the invalid error type `box.session.su()` raises for a not found user
  - was: `SystemError`
  - now: `ClientError`

### Building from sources

- Fix for GCC 4.8.5, which is default version on CentOS 7 (gh-4438).
- Fix OpenSSL linking problems on FreeBSD (gh-4490).
- Fix linking problems on Mac OS when several toolchains are in PATH (gh-4587).
- Fix GCC 9 warning on strncpy() (gh-4515).
- Fix build on Mac with gcc and XCode 11 (gh-4580).
- Fix LTO warnings that were treated as errors in a release build (gh-4512).

## Tarantool 2.2.3

Release: v. 2.2.3 Date: 2020-04-20 Tag: 2.2.3-1-g98ecc90

### Общие сведения

2.2.3 is the last stable version of the 2.2 release series. The label stable means we have all planned features implemented and we see no high-impact issues.

This release resolves roughly 34 issues since the latest stable version. There may be bugs in less common areas, please feel free to file an issue at GitHub.

Please note, this release contains **no new features**.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the box.schema.upgrade() procedure to unlock all the new features of the 2.x series.

### Исправленные ошибки

#### Core

- *fiber.storage* is cleaned between requests, and can be used as a request-local storage. Previously `fiber.storage` could contain some old values in the beginning of an iproto request execution, and it needed to be nullified manually. Now the cleanup is unneeded (gh-4662).
- `tuple`/`space`/`index`:`update()`/`upsert()` were fixed not to turn a value into an infinity when a float value was added to or subtracted from a float column and exceeded the float value range (gh-4701).
- Fix potential execution abort when operating the system runs under heavy memory load (gh-4722).
- Make RTREE indexes handle the out of memory error: before this fix, OOM during the recovery of an RTREE index resulted in segmentation fault (gh-4619).
- Fix the error message returned on using an already dropped sequence (gh-4753).
- Add cancellation guard to avoid WAL thread stuck (gh-4127).
- Fix execution abort when `memtx_memory` and `vinyl_memory` are set to more than 4398046510080 bytes. Now an error message is returned (gh-4705).

#### Репликация

- Fix rebootstrap procedure not working in case replica itself is listed in `box.cfg.replication` (gh-4759).
- Fix possible user password leaking via replication logs (gh-4493).
- Refactor vclock map to be exactly 4 bytes in size to fit all 32 replicas regardless of the compiler used (https://github.com/tarantool/tarantool/commit/e5679980aa5f813553a95ab7d31f111dd0893df6).
- Fix crash when the replication applier rollbacks a transaction (gh-4730, (gh-4776).
- Local space operations are now counted in 0th vclock component. Every instance may have its own 0-th vclock component not matching others'. Local space operations are not replicated at all, even as NOPs (gh-4114).
- Gc consumers are now ordered by their vclocks and not by vclock signatures. Only the WALS that contain no entries needed by any of the consumers are deleted (gh-4114).

### Lua

- json: `:decode()` doesn't spoil instance's options with per-call ones (when it is called with the second argument) (gh-4761).
- Handle empty input for `uri.format()` properly (gh-4779).
- `os.environ()` is now changed when `os.setenv()` is called (gh-4733).
- `netbox.self:call/eval()` now returns the same types as `netbox_connection:call/eval`. Previously it could return a tuple or `box.error` cdata (gh-4513).
- `box.tuple.*` namespace is cleaned up from private functions. *box.tuple.is()* description is added to documentation (gh-4684).
- `tarantoolctl rocks search`: fix the `--all` flag (gh-4529).
- `tarantoolctl rocks remove`: fix the `--force` flag (gh-3632).
- libev: backport fix for listening for more then 1024 file descriptors on Mac OS (gh-3867).

### SQL

- Fix `box.stat()` behavior: now it collects statistics on the `PREPARE` and `EXECUTE` methods as expected (gh-4756).
- The inserted values are inserted in the order in which they are given in case of INSERT into space with autoincrement (gh-4256).

### HTTP client

- When building Tarantool with bundled `libcurl`, link it with the `c-ares` library by default (gh-4591).

### LuaJIT

- `__pairs`/`__ipairs` metamethods handling is removed since we faced the issues with the backward compatibility between Lua 5.1 and Lua 5.2 within Tarantool modules as well as other third party code (gh-4770).
- Introduce `luajit-gdb.py` extension with commands for inspecting LuaJIT internals. The extension obliges one to provide gdbinfo for `libluajit`, otherwise loading fails. The extension provides the following commands:
  - `lj-arch` dumps values of LJ_64 and LJ_GC64 macro definitions
  - `lj-tv` dumps the type and GCobj info related to the given TValue
  - `lj-str` dumps the contents of the given GCstr
  - `lj-tab` dumps the contents of the given GCtab
  - `lj-stack` dumps Lua stack of the given lua_State
  - `lj-state` shows current VM, GC and JIT states
  - `lj-gc` shows current GC stats
- Fix string to number conversion: current implementation respects the buffer length (gh-4773).

- **“FFI sandwich”(*) detection is introduced. If sandwich is detected** while trace recording the recording is aborted. The sandwich detected while mcode execution leads to the platform panic.
- luaJIT_setmode call is prohibited while mcode execution and leads to the platform panic.

(*) The following stack mix is called FFI sandwich:

Lua-FFI -> C routine -> Lua-C API -> Lua VM

This sort of re-entrancy is explicitly not supported by LuaJIT compiler. For more info see gh-4427.

#### Vinyl

- Fix assertion fault due to triggered dump process during secondary index build (gh-4810).

#### Разное

- Fix crashes at attempts to use `-e` and `-l` command line options concatenated with their values, like this: `-eprint(100)` (gh-4775).
- Fix inability to upgrade from 2.1 if there was an automatically generated sequence (gh-4771).
- Update `libopenssl` version to 1.1.1f since the previous one was EOLed (gh-4830).

#### Building from sources

- Fix build of the `decNumber` library under OSX (gh-4580).
- Update the `decNumber` library to silence the build warning produced on too long integer constant (`https://github.com/tarantool/tarantool/commit/aab03a735c7a215b4371ef834f7d08432b1bf0f7`).
- Fix static build (`-DBUILD_STATIC=ON`) when `libunwind` depends on `liblzma` (gh-4551).

### Tarantool 2.2.2

Release: v. 2.2.2 Date: 2019-12-31 Tag: 2.2.2-0-g0a577ff

#### Общие сведения

2.2.2 is the first stable version of the 2.2 release series. The label stable means we have all planned features implemented and we see no high-impact issues.

This release resolves roughly 75 issues since the latest beta version. There may be bugs in less common areas, please feel free to file an issue at GitHub.

### Совместимость

Tarantool 2.x is backward compatible with Tarantool 1.10.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 2.x series.

### Изменения или добавления функциональности

There are changes labeled with **[Breaking change]**. It means that the old behaviour was considered error-prone and therefore changed to protect users from unintended mistakes. However, there is a little probability that someone can lean on the old behaviour, and this label is to bring attention to the things that have been changed.

#### Core

- **[Breaking change]** Drop `rows_per_wal` `box.cfg()` option in favor of `wal_max_size` (gh-3762)

#### Lua

- **[Breaking change]** json and msgpack serializers now raise an error when a depth of data nesting exceeds `encode_max_depth` option value. The default value of the option is increased from 32 to 128. `encode_deep_as_nil` option was added to give ability to set the old behaviour back (gh-4434). Note: Those options can be set using `json.cfg({<...>})` or `msgpack.cfg({<...>})`. Note: box data modification functions (`insert`, `replace`, `update`, `upsert`) follows options of default msgpack serializer instance and now they will raise an error by default on too nested data rather then cut them silently. This behaviour can be configured using `msgpack.cfg({<...>})`. Note: `box.tuple.new()`, `space:update()`, `space:upsert()` and several other functions did not follow `encode_max_depth` option, now they do (see also 'Bug fixes'). Note: `json.cfg` and `msgpack.cfg` tables did not updated when an option is changed, now they show actual values (see also 'Bug fixes').
- Show line and column in json.decode() errors (gh-3316)
- Exit gracefully when a main script throws an error: notify systemd, log the error (gh-4382)
- key_def: accept both `field` and `fieldno` in `key_def.new(<...>)` (gh-4519). Originally `key_def.new(<...>)` accepts only `fieldno` to allow creation with `<index_object>.parts` as argument. However index definition format (`<space_object>.create_index(<...>)`) is different and requires `field`. Now both are supported.
- Enable `__pairs` and `__ipairs` metamethods from Lua 5.2 (gh-4560). We still conform Lua 5.1 API, which is not always compatible with Lua 5.2. The change is only about those metamethods.

#### Разное

- tarantoolctl: allow to start instances with delayed box.cfg{} (gh-4435)
- Add package builds and deployment for the following Linux distros:
  - Ubuntu 19.10 Eoan Ermine (gh-4583)
  - CentOS 8 (gh-4543)
  - Fedora 31 (gh-4612)

#### Исправленные ошибки

### SQL

- Modify type of a binding value in a query response metainformation: always return INTEGER rather then UNSIGNED, even for positive values. This is necessary for consistency with integer literal types. c5246686.
- Reuse noSQL way to compare floating point values with integral ones. This allows to handle corner cases like `SELECT 18446744073709551615.0 > 18446744073709551615` uniformly. 12431ed4.
- Create or alter of a table with a foreign key may lead to wrong bytecode generation that may cause a crash or wrong result (gh-4495)
- Allow to update a scalar value using SQL in a space that was created from Lua and contains 'array', 'map' or 'any' field (gh-4189). Note: Tarantool/SQL provides operations on scalar types and does not support 'array' and 'map' per se.
- `INDEXED BY` clause now obligates the query planner to choose provided index, 411be0f0
- Fix dirty memory access when constructing query plan involving search of floating point value in index over integer field (gh-4558)

### Core

- Detect a new invalid json path case (gh-4419)
- Randomize the next checkpoint time also after a manual `box.snapshot()` (gh-4432)
- Fix memory leak in call / eval in the case when a transaction is not committed (gh-4388)
- Eliminate warning re 'strip_core' box.cfg option on MacOS and FreeBSD (gh-4464)
- The msgpack serializer that is under `box.tuple.new()` (called tuple serializer) did not reflect options set by `msgpack.cfg({<...>})`, part of (gh-4434). Aside of `box.tuple.new()` behaviour itself, it may affect tuple:frommap(), methods of key_def Lua module, tuple and table merger sources, net.box results of `:select()` and `:execute()` calls, xlog Lua module.
- box's `update` and `upsert` now follow `msgpack.cfg({encode_max_depth = <...>}` option, part of (gh-4434)
- fiber: make sure the guard page is created, refuse to create a new fiber otherwise (gh-4541). It is possible in case of heavy memory pressure, say, when there is no resources to split VMAs.
- recovery: build secondary indices in the hot standby mode without waiting till the main instance termination (gh-4135)
- Fix error message for incorrect return value of functional index extractor function (gh-4553)

  - Was: «Key format doesn't match one defined in functional index '' of space '': supplied key type is invalid: expected boolean'
  - Now: "<...>: expected array"

- JSON path index did ignore is_nullable property when a space had a format (gh-4520)
- Forbid `00000000-0000-0000-0000-000000000000` as `replicaset_uuid` and `instance_uuid` `box.cfg({<...>})` options value, (gh-4282). It did not work as expected: the nil UUID was treated as absence of a value.
- Update cache of universe privileges without reconnect (gh-2763)
- net.box: fix memory leak in `net_box:connect(<URI>)` (gh-4588)
- net.box: don't fire `on_connect` trigger at schema update (gh-4593). Also don't fire `on_disconnect` trigger if a connection never entered into 'active' state (e.g. when a first schema fetch is failed).
- func: fix use after free on function unload, 64f4d06a
- Fix bootstrap.snap file in order to overcome the following warning, (gh-4510) > xlog.c:1934 E> can't open tx: bootstrap: has some data after eof marker at 5902
- Don't destroy a session until `box.session.on_disconnect(<...>)` triggers will be finished (gh-4627). This means that, say, `box.session.id()` can be safely invoked from the `on_disconnect` trigger. Before this change `box.session.id()` returns garbage (usually 0) after yield in the `on_disconnect` trigger. Note: tarantool/queue module is affected by this problem in some scenarious. It is especially suggested to update tarantool at least to this release if you're using this module.
- func: `box.schema.func.drop(<..>)` did not unload unused modules (gh-4648). Also `box.schema.func.create(<..>)` did load of a module again even when another function from the module is loaded.
- Encode Lua number -2^63 as integer in msgpack.encode() and box's functions (gh-4672)

#### Репликация

- Stop relay on subscribe error (gh-4399)
- Set `last_row_time` to `now` in `relay_new` and `relay_start` (gh-4431)
- Do not abort replication on ER_UNKNOWN_REPLICA (gh-4455)
- Enter orphan mode on manual replication configuration change (gh-4424)
- Disallow bootstrap of read-only masters (gh-4321)
- Prefer to bootstrap a replica from a fully bootstrapped instance rather than currently bootstrapping one (gh-4527). This change enables the case when two nodes (B, C) are being bootstrapped simultaneously using the one that is already bootstrapped (A), while A is configured to replicate from {B, C} and B from {A, C}.
- Return immediately from `box.cfg{<...>}` when an instance is reconfigured with `replication_connect_quorum = 0` (gh-3760) This change also fixes the behaviour of reconfiguration with non-zero `replication_connect_quorum`: `box.cfg{<...>}` returns immediately regardless of whether connections to upstreams are established.
- Apply replication `box.cfg({<...>})` settings in a strict order (gh-4433)
- Auto reconnect a replica if password is invalid (gh-4550)
- `box.session.su(<username>)` now reports an error correctly for `<username>` longer then BOX_NAME_MAX, which is 65000, 43e29191 Was: 'C++ exception' Now: 'name length <...> is greater than BOX_NAME_MAX'

- Use empty password when an URI in `box.cfg{replication = <...>}` is like `login@host:port` (gh-4605). The behaviour match net.box's one now. Explicit `login:@host:port` was necessary before, otherwise a replica shows the following error: > Missing mandatory field 'tuple' in request
- Fix segfault during replication configuration (`box.cfg{replication = <...>}` call) (gh-4440, gh-4576, gh-4586, gh-4643)

### Lua

- Fix segfault on `ffi.C._say()` without filename (gh-4336)
- Fix `pwd.getpwall()` and `pwd.getgrall()` hang on CentOS 6 and FreeBSD 12 (gh-4428, gh-4447)
- json.encode() now follows `encode_max_depth` option for arrays that leads to a segfault on recursive Lua tables with numeric keys (gh-4366)
- fio.mktree() now reports an error for existing non-directory file (gh-4439)
- Update `json.cfg` and `msgpack.cfg` tables when an option is changed, part of (gh-4434)
- Fix handling of a socket read error on the console client (`console.connect(<URI>)` or `tarantoolctl connect/enter <...>`), b0b19992
- Handle 'not enough memory' gracefully when it is raised from `lua_newthread()` (gh-4556). There are several places where a new Lua thread is created:
  - Start execution a Lua function call or an eval request (from a binary protocol, SQL or with `box.func.<...>:call()`).
  - Create of a new fiber.
  - Start execution of a trigger.
  - Start of encoding into a YAML format (`yaml.encode()`).
- Fix stack-use-after-scope in json.decode() (gh-4637)

### HTTP client

- Use bundled `libcurl` rather than system-wide by default, (gh-4318, gh-4180, gh-4288, gh-4389, gh-4397). This closes several known problems that were fixed in recent libcurl versions, including segfaults, hangs, memory leaks and performance problems.
- Disable verbose mode when `{verbose = false}` is passed, 5f3d9015
- Fix assertion fail after curl write error (gh-4232)

### Console Lua output

The new Lua output format is still in the alpha stage and has known flaws, but we are working to make it rich and stable.

- Output `box.NULL` as `"box.NULL"` rather then `"cdata<void *>: NULL"`, part of (gh-3834) (in quotes for now, yes, due to (gh-4585)
- Add semicolon (`;`) as responses delimiter (EOS, end of stream/statement), analogue of YAMLs end-of-document (`...`) marker. This is vital for remote clients to determine an end of a particular response, part of (gh-3834).

- Fix hang in the console client (`console.connect(<URI>)` or `tarantoolctl connect/enter <...>`) after `\set output lua[,block]` command, part of (gh-3834). In order to overcome it two changes were made:
  - Parse `\set output lua[,block]` command on a client prior to sending it to a server, store current responses delimiter (EOS) and use it to determine end of responses.
  - Send `\set output <...>` command with a default output mode when establishing a connection (it is matter if different default modes are set).
- Provide ability to get or set current responses delimiter using `console.eos([<...>])`, part of (gh-3834)

### LuaJIT

- Fix fold machinery misbehaves (gh-4376)
- Fix for `debug.getinfo(1,'>S')` (gh-3833)
- Fix `string.find` recording (gh-4476)
- Fixed 'Data segment size exceeds process limit' error on FreeBSD/x64: do not change resource limits when it is not necessary (gh-4537)
- fold: keep type of emitted CONV in sync with its mode, LuaJIT#524 This fixes the following assertion fail: > asm_conv: Assertion `((IRType)((ir->t).irt & IRT_TYPE)) != st' failed

### Разное

- Support `systemd`'s NOTIFY_SOCKET on OS X (gh-4436)
- Fix linking with static `openssl` library (gh-4437)
- Get rid of warning re empty `NOTIFY_SOCKET` variable (gh-4305)
- rocks: fix 'invalid date format' error when installing a packed rock (gh-4481)
- Remove libyaml from rpm/deb dependencies, because we use bunbled version of libyaml for the packages (since 2.2.1) (gh-4442)
- Fix boolean CLI options handling in `tarantoolctl cat <...>`, such as `--show-system` (gh-4076)
- Fix segfault (out of bounds access) when unwinding error occurs at backtrace printing (gh-4636). Backtrace is printed on SIGFPE and SIGSEGV signal or when LuaJIT find itself in unrecoverable state (lua_atpanic()).

### Building from sources

- Fix for GCC 4.8.5, which is default version on CentOS 7 (gh-4438)
- Fix OpenSSL linking problems on FreeBSD (gh-4490)
- Fix linking problems on Mac OS when several toolchains are in PATH (gh-4587)
- Fix GCC 9 warning on strncpy() (gh-4515)

### Tarantool 2.2.1

Release: v. 2.2.1. Release type: beta. Release date: 2019-08-02.

#### Overview

This is a *beta* version of the 2.2 series. The label «beta» means we have no critical issues and all planned features are there.

The goal of this release is to introduce new indexing features, extend SQL feature set, and improve integration with the core.

#### Compatibility

Tarantool 2.x обратно совместим с Tarantool 1.10.x в том, что касается структуры бинарных данных, клиент-серверного протокола и протокола репликации. *Обновление* можно произвести с помощью процедуры `box.schema.upgrade()`.

#### Functionality added or changed

- (SQL) ALTER now allows to add a constraint:

```
CREATE TABLE t2 (id INT PRIMARY KEY);
ALTER TABLE t2 ADD CONSTRAINT ck CHECK(id  > 0);
```

- (SQL) CHECK constraints are validated during DML operations performed from the Lua land:

```
s = box.schema.space.create('withdata')
pk = s:create_index('pk')
s:format({{'idx', 'number'}})
s:create_check_constraint('le10', '"idx" < 10')
```

```
tarantool> s:insert({11})
---
- error: 'Check constraint failed ''le10'': "idx" < 10'
...
```

- (SQL) New *SQL types* introduced: VARBINARY, UNSIGNED, and `BOOLEAN`.
- (SQL) CREATE TABLE statement (and all other data definition statements) are now truly transactional.
- (SQL) SQL now uses Tarantool diagnostics API to set errors, so error reporting now provides an error code in addition to error message.
- (SQL) Multiple improvements to the type system to make it more consistent.
- (SQL) Added aliases for LENGTH() from ANSI SQL: CHAR_LENGTH() and CHARACTER_LENGTH().
- (SQL) It is possible to use HAVING without GROUP BY.
- (Server) New fixed point type (DECIMAL) introduced to Tarantool:

```
decimal = require('decimal')
tarantool> a = decimal.new('123.456789')
---
...
tarantool> decimal.precision(a)
---
- 9
...
tarantool> decimal.scale(a)
---
- 6
...
tarantool> decimal.round(a, 4)
---
- '123.4568'
...
```

- (Server) Multikey index support:

```
-- Multikey indexes (for memtx tree & vinyl);
-- cannot be primary; may be non-unique
s = box.schema.space.create('clients', {engine = 'vinyl'})
pk = s:create_index('pk')
phone_type = s:create_index('phone_type', {
    unique = false,
    parts = {{'[3][*].type', 'str'}}})

s:insert({1, 'James',
         {{type = 'home', number = '999'},
          {type = 'work', number = '777'}
         }})
s:insert({2, 'Bob',
          {{type = 'work', number = '888'}}})
s:insert({3, 'Alice', {{type = 'home', number = '333'}}})
```

```
tarantool> phone_type:select('work')
---
- - [1, 'James', [{'type': 'home', 'number': '999'},
                  {'type': 'work', 'number': '777'}]]
  - [2, 'Bob', [{'type': 'work', 'number': '888'}]]
...
```

- (Server) Now it is possible to make functions persistent:

```
box.schema.func.create('summarize',
                       {body = [[function(a,b) return a+b end]],
                       is_deterministic = true})
```

```
tarantool> box.func.summarize
- aggregate: none
  returns: any
  exports:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    lua: true
    sql: false
  id: 66
  is_sandboxed: false
      setuid: false
  is_multikey: false
  is_deterministic: true
  body: function(a,b) return a+b end
  name: summarize
  language: LUA

tarantool> box.func.summarize:call({1, 2})
---
- 3
...
```

- (Server) Functional indexes implemented:

```
-- Functional multikey indexes: define is_multikey = true
-- in function definition and return a table of keys from function
lua_code = [[function(tuple)
                local address = string.split(tuple[2])
                local ret = {}
                for _, v in pairs(address) do table.insert(ret, {utf8.upper(v)}) end
                return ret
             end]]
box.schema.func.create('addr_extractor', {body = lua_code,
                                          is_deterministic = true,
                                          is_sandboxed = true,
                                          opts = {is_multikey = true}})
s = box.schema.space.create('withdata')
pk = s:create_index('name', {parts = {1, 'string'}})
idx = s:create_index('addr', {unique = false, func = box.func.addr_extractor.id,␣
↪parts = {{1, 'string', collation = 'unicode_ci'}}})

s:insert({"James", "SIS Building Lambeth London UK"})
s:insert({"Sherlock", "221B Baker St Marylebone London NW1 6XE UK"})
```

```
tarantool>  idx:select('Sis')
---
- - ['James', 'SIS Building Lambeth London UK']
...
```

- Partial core dumps, which are now on by default. It is now possible to avoid dumping tuples at all during core dump.
- Data definition statements, such as create or alter index, which do not yield, can now be used in a transaction. This in practice includes all statements except creating an index on a non-empty space, or changing a format on a non-empty space.
- It is now possible to set a sequence not only for the first part of the index:

```
s.index.pk:alter{sequence = {field = 2}}
```

- Allow to call `box.session.exists()` and `box.session.fd()` without any arguments.
- New function introduced to get an index key from a tuple:

```
s = box.schema.space.create('withdata')
pk = s:create_index('pk')
sk = s:create_index('sk', {parts = {
      {2, 'number', path = 'a'},
      {2, 'number', path = 'b'}}})
s:insert{1, {a = 1, b = 1}}
s:insert{2, {a = 1, b = 2}}
s:insert{3, {a = 3, b = 3}}
sk:select(2)

key_def_lib = require('key_def')
key_def = key_def_lib.new(pk.parts)
for _, tuple in sk:pairs({1}) do
    local key = key_def:extract_key(tuple)
    pk:delete(key)
end
s:select()
```

- (Engines) New protocol (called *SWIM*) implemented to keep a table of cluster members.
- (Engines) Removed yields from Vinyl DDL on commit triggers.
- (Engines) Improved performance of SELECT-s on memtx spaces. The drawback is that now every memtx-tree tuple consumes extra 8 bytes for a search hint.
- (Engines) Indexes of memtx spaces are now built in background fibers. This means that we do not block the event loop during index build anymore.
- Replication applier now can apply transactions which were concurrent on the master concurrently on replica. This dramatically improves replication peak performance, from ~50K writes per second to 200K writes per second and higher on a single instance.
- Transaction boundaries introduced to replication protocol. This means that Tarantool replication is now transaction-safe, and also reduces load on replica write ahead log in case the master uses a lot of multi-statement transactions.
- Tuple access by field name for `net.box`:

```
box.cfg{listen = 3302}
box.schema.user.grant('guest','read, write, execute', 'space')
box.schema.user.grant('guest', 'create', 'space')
box.schema.create_space("named", {format = {{name = "id"}}})
box.space.named:create_index('id', {parts = {{1, 'unsigned'}}})
box.space.named:insert({1})

require('net.box').connect('localhost', 3302).space.named:get(1).id
```

- Cluster id check is now the slave's responsibility.
- It is now possible to set the output format to Lua instead of YAML in the *interactive console*.
- Multiple new collations added. New collations follow this naming pattern:

```
unicode_<locale>_<strength>
```

Three strengths are used:

- Primary - «s1"
- Secondary - «s2»
- Tertiary - «s3»

The following list contains so-called «stable» collations – the ones whose sort order doesn't depend on the ICU version:

```
unicode_am_s3
unicode_fi_s3
unicode_de__phonebook_s3
unicode_haw_s3
unicode_he_s3
unicode_hi_s3
unicode_is_s3
unicode_ja_s3
unicode_ko_s3
unicode_lt_s3
unicode_pl_s3
unicode_si_s3
unicode_es_s3
```

- New function `utime()` introduced to the `fio` module.
- *Merger* for tuples streams added.

### Tarantool 2.1.2 and earlier

Release: https://github.com/tarantool/tarantool/releases

**Версия 2.1.2**

Release type: stable. Release date: 2019-04-05.

Release: v. 2.1.2.

This is the first *stable* release in the 2.x series.

The goal of this release is to significantly extend SQL support and increase stability.

Изменения или добавления функциональности:

- **(SQL)** `box.sql.execute()` **replaced with** *box.execute()*. It now works just like `netbox.execute()`: returns result set metadata, row count, etc. E.g.:

```
box.execute("CREATE TABLE person(id INTEGER PRIMARY KEY, birth_year INT)")
---
- row_count: 1
...
box.execute("SELECT birth_year FROM person")
---
- metadata:
  - name: birth_year
    type: INTEGER
  rows:
  - [1983]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  - [1984]
...
```

- (SQL) Type system was *significantly refactored*.
- **(SQL) There are cases in SQL when it is possible to do Tarantool's** update operation for UPDATE statement, instead of doing delete + insert. However, there are cases where SQL semantics is too complex. E.g.:

```
CREATE TABLE file (id INT PRIMARY KEY, checksum INT);
INSERT INTO stock VALUES (1, 3),(2, 4),(3,5);
CREATE UNIQUE INDEX i ON file (checksum);
SELECT * FROM file;
-- [1, 3], [2, 4], [3, 5]
UPDATE OR REPLACE file SET checksum = checksum + 1;
SELECT * FROM stock;
-- [1, 4], [3, 6]
```

  I.e. [1, 3] tuple is updated as [1, 4] and have replaced tuple [2, 4]. This logic is implemented by preventive tuple deletion from all corresponding indexes in SQL.

- **(SQL) Now SQL's integer type is stored as integer in space's format.** It was stored as scalar before, which made comparisons slow.
- **(SQL) It is now possible to define a constraint** *within column definition*. E.g.:

```
CREATE TABLE person (id INT PRIMARY KEY, age INT, CHECK (age > 10));
```

- **(SQL) Syntax for the pragma `pragma index_info` is now unified with `table_info`.** E.g. to get information on index `age_index` of table `person` you can write:

```
pragma index_info(person.age_index);
```

- (Server) It is now possible to index a field specified using JSON. E.g.:

```
person = box.schema.create_space("person")
name_idx = person:create_index('name', {parts = {{'[2]fname', 'str'}, {
↪'[2]sname', 'str'}}})
person:insert({1, {fname='James', sname='Bond'}, {town='London', country='GB
↪', organization='MI6'}})
```

- (Server) In case of out of space event, Tarantool is now allowed to delete backup WAL files not needed for recovery from the last checkpoint.
- (Server) Add support for *tarantoolctl rocks pack / unpack* subcommands. The subcommands are used to create / deploy binary rock distributions.
- (Server) `string.rstrip` and `string.lstrip` should accept symbols to strip. Add optional „chars“ parameter for specifying the unwanted characters. E.g.:

```
local chars = "#\0"
str = "##Hello world!#"
print(string.strip(str, chars)) -- "Hello world!"
```

- **(Server) *on_shutdown* trigger added.** It may be set in a way similar to `space:on_replace` triggers:

```
box.ctl.on_shutdown(new_trigger, old_trigger)
```

- **(Server)** ***on_schema_init*** **trigger added.** It may be set before the first call to `box.cfg()` and is fired during `box.cfg()` before user data recovery start. To set the trigger, say:

```
box.ctl.on_schema_init(new_trig, old_trig)
```

- **(Server) A new option for the snapshot daemon,** *box.cfg.checkpoint_wal_threshold*, allows to limit the maximum disk size of maintained WALs. Once the configured threshold is exceeded, the WAL thread notifies the che ckpoint daemon that it's time to make a new checkpoint and delete old WAL files.
- **(Server) New types of** ***privileges*** – to create, alter and drop space – were introduced. In order to create, drop or alter space or index, you should have a corresponding privilege. E.g.:

```
box.schema.user.create("optimizer", { password  = 'secret' })
box.schema.user.grant("optimizer", "alter", "space")
person = box.schema.space.create("person")
box.session.su("optimizer")
i = s:c reate_index("primary") -- success
s:insert{1} -- fail
s:select{} -- fail
s:drop() -- fail
```

  Notice the incompatible change: Tarantool 1.10 requires read/write/execute privileges on an object to allow create, drop or alter. These privileges are no longer sufficient in 2.1. To remedy the problem, Tarantool 2.1 automatically grants create/drop/alter privileges on an object if a user has read/write/execute privileges on it during schema upgrade. But old scripts may stop working if read/write/execute is granted **after** schema upgrade.

  Additionally, create/drop/alter privileges are already supported in 1.10, which also supports the old semantics of read/write/execute. You are encouraged to grant new privileges in 1.10 before upgrade and modify your scripts.

**Версия 2.1.1**

Release type: beta. Release date: 2018-11-14.

Release: v. 2.1.1.

This release resolves all major bugs since 2.0.4 alpha and extends Tarantool's SQL feature set.

**Версия 2.0.4**

Тип версии: альфа. Дата выхода: 2018-02-15.

Release: v. 2.0.4.

This is a successor of the 1.8.x releases. It improves the overall stability of the SQL engine and has some new features.

Изменения или добавления функциональности:

- Added support for SQL collations by incorporating libICU character set and collation library.
- IPROTO interface was extended to support SQL queries.
- `net.box` subsystem was extended to support SQL queries.
- Enabled `ANALYZE` statement to produce correct results, necessary for efficient query plans.
- Enabled savepoints functionality. `SAVEPOINT` statement works w/o issues.

- Enabled `ALTER TABLE ... RENAME` statement.
- Improved rules for identifier names: now fully consistent with Lua frontend.
- Enabled support for triggers; trigger bodies now persist in Tarantool snapshots and survive server restart.
- Significant performance improvements.

### Tarantool 1.10.15

Released on 2023-02-20.

- Release: v. 1.10.15.
- Tag: `1.10.15`.

#### Overview

1.10.15 is the next stable release in the *long-term support (LTS) version* 1.10.x release series.

The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release introduces 2 improvements and resolves roughly 8 issues since the *1.10.14* version.

#### Compatibility

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 1.10.x series.

#### Functionality added or changed

#### Build

- OpenSUSE 15.1 and 15.2 are no longer supported.
- Updated libcurl to version 7.87.0 (gh-8150).

#### Bugs fixed

#### Core

- Fixed a bug when fields could be removed from a table stored in a variable when a logging function was called on this variable (for example, `log.info(a)`) (gh-3853).
- Fixed a logging bug: when logging tables with fields that have reserved internal names (such as `pid`) in the plain log format, such fields weren't logged (gh-3853).
- Added the `message` field when logging tables without such field in the JSON log format (gh-3853).
- Fixed an assertion on malformed JSON message written to the log (gh-7955).

### Vinyl

- Fixed a bug that could result in `select()` skipping an existing tuple after a rolled back `delete()` (gh-7947).

### LuaJIT

Backported patches from vanilla LuaJIT trunk (gh-7230). In the scope of this activity, the following issues have been resolved:

- Fix overflow check in `unpack()` optimized by a compiler.
- Fix recording of `tonumber()` with cdata argument for failed conversions (gh-7655).
- Fix concatenation operation on cdata. It always raises an error now.
- Fix `io.close()` for already closed standard output.
- Fix trace execution and stitching inside vmevent handler (gh-6782).
- Fixed `emit_loadi()` on x86/x64 emitting xor between condition check and jump instructions.
- Fix stack top for error message when raising the OOM error (gh-3840).
- Disabled math.modf compilation due to its rare usage and difficulties with proper implementation of the corresponding JIT machinery.
- Fixed inconsistent behaviour on signed zeros for JIT-compiled unary minus (gh-6976).
- Fixed `IR_HREF` hash calculations for non-string GC objects for GC64.
- Fixed assembling of type-check-only variant of `IR_SLOAD`.
- Fixed the Lua stack dump command (`lj-stack`) not working on Python 2. Previously, it used arguments unpacking within the list initialization, which is not supported in Python 2 (gh-7458).

Backported patches from vanilla LuaJIT trunk (gh-8069). In the scope of this activity, the following issues have been resolved:

- Fixed loop realigment for dual-number mode
- Fixed os.date() for wider libc strftime() compatibility.

### Fiber

- Fixed the assertion fail in `cord_on_yield` (gh-6647).

### Log

- Fixed an incorrect facility value in syslog on Alpine (gh-8269).

#### Build

- Fixed `-Werror` build fail on Clang 15 (gh-8110).

### Tarantool 1.10.14

Released on 2022-08-08.

- Release: v. 1.10.14.
- Tag: `1.10.14`.

#### Overview

1.10.14 is the next stable release in the *long-term support (LTS) version* 1.10.x release series.

The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release introduces 10 improvements and resolves roughly 20 issues since the *1.10.13* version.

#### Compatibility

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 1.10.x series.

#### Functionality added or changed

#### Build

- Fedora 35 is now supported (gh-6692).
- Fedora 36 is now supported.
- Ubuntu 22.04 (Jammy Jellyfish) is now supported.
- Fedora 30, 31, 32, and 33 are no longer supported.
- Ubuntu 21.04 (Hirsute Hippo) is no longer supported.
- Updated OpenSSL used for static builds to version 1.1.1n (gh-6947).
- Updated OpenSSL used for static builds to version 1.1.1q.
- Updated libcurl to version 7.83.0 (gh-6029).
- Updated libcurl to version 7.84.0.
- Updated libyaml to the version with fixed stack overflows.

### Bugs fixed

#### Core

- Fixed a memory leak in the interactive console (gh-6817).
- Fixed an assertion fail when passing a tuple without the primary key fields to a `before_replace` trigger. Now the tuple format is checked before the execution of `before_replace` triggers and after each of them (gh-6780).
- Now inserting a tuple with a wrong `id` field into the `\_priv` space returns the correct error (gh-6295).
- Fixed a bug that was making all fibers created with `fiber_attr_setstacksize()` leak until the thread exit. Their stacks also leaked except when `fiber_set_joinable(..., true)` was used.
- Fixed a crash that happened when Tarantool was launched with multiple `-e` or `-l` options without spaces between the options and their values (gh-5747).
- Fixed the usage of `box.session.peer()` in `box.session.on_disconnect()` triggers. Now it's safe to assume that `box.session.peer()` returns the address of the disconnected peer, not nil, as it used to (gh-7014).
- Fixed a bug in the sequence cache that could result in an error creating a new sequence (gh-5306).

#### Vinyl

- Immediate removal of compacted run files created after the last checkpoint optimization now works for the initial JOIN stage of a replica (gh-6568).
- Fixed a crash during the recovery of a secondary index in case the primary index contains incompatible phantom tuples (gh-6778).
- Fixed a bug in the vinyl upsert squashing optimization that could lead to a segmentation fault error (gh-5080).
- Fixed a bug in the vinyl read iterator that could result in a significant performance degradation of range select requests in the presence of an intensive write workload (gh-5700).

#### Replication

- Fixed replicas failing to bootstrap when the master has just restarted (gh-6966).

#### LuaJIT

- Fixed the top part of Lua stack (red zone, free slots, top slot) unwinding in the `lj-stack` command.
- Added the value of `g->gc.mmudata` field to `lj-gc` output.
- Fixed a bug with `string.char()` builtin recording when no arguments are provided (gh-6371, gh-6548).
- Actually made JIT respect the `maxirconst` trace limit while recording (gh-6548).
- Backported patches from vanilla LuaJIT trunk (gh-6548, gh-7230). In the scope of this activity, the following issues have been resolved:
  - Fixed emitting for fuse load of constant in GC64 mode (gh-4095, gh-4199, gh-4614).
  - Now initialization of zero-filled struct is compiled (gh-4630, gh-5885).

- Actually implemented `maxirconst` option for tuning JIT compiler.
- Fixed JIT stack of Lua slots overflow during recording for metamethod calls.
- Fixed bytecode dump unpatching for JLOOP in up-recursion compiled functions.
- Fixed FOLD rule for strength reduction of widening in cdata indexing.
- Fixed `string.char()` recording without arguments.
- Fixed `print()` behaviour with the reloaded default metatable for numbers.
- `tonumber("-0")` now saves the sign of number for conversion.
- `tonumber()` now gives predictable results for negative non-base-10 numbers.
- Fixed write barrier for `debug.setupvalue()` and `lua_setupvalue()`.
- `jit.p` now flushes and closes output file after run, not at program exit.
- Fixed `jit.p` profiler interaction with GC finalizers.
- Fixed the case for partial recording of vararg function body with the fixed number of result values in with `LJ_GC64` (i.e. `LJ_FR2` enabled) (gh-7172).
- Fixed handling of errors during trace snapshot restore.

#### Box

- Added the check of the iterator type in the `select`, `count`, and `pairs` methods of the index object. Iterator can now be passed to these methods directly: `box.index.ALL`, `box.index.GT`, and so on (gh-6501).

#### Recovery

- With the `force_recovery` cfg option, Tarantool is now able to boot from `snap`/`xlog` combinations where `xlog` covers changes committed both before and after the `snap` was created. For example, `0...0.xlog` that covers everything up to vclock `{1: 15}` and `0...09.snap` corresponding to vclock `{1: 9}` (gh-6794).

### Tarantool 1.10.13

Released on 2022-04-25.

- Release: v. 1.10.13.
- Tag: `1.10.13`.

### Overview

1.10.13 is the next stable release in the *long-term support (LTS) version* 1.10.x release series.

The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release introduces 1 improvement and resolves roughly 13 issues since the *1.10.12* version.

### Compatibility

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 1.10.x series.

### Functionality added or changed

### Build

- Support fedora-35 build (gh-6692).

### Bugs fixed

### Core

- Fixed memory leak in interactive console (gh-6817).
- Fixed an assertion fail when passing tuple without primary key fields to `before_replace` trigger. Now tuple format is checked before the execution of `before_replace` triggers and after each one (gh-6780).
- Now inserting a tuple with the wrong `id` field into the `_priv` space returns the correct error (gh-6295).
- Fixed a bug due to which all fibers created with `fiber_attr_setstacksize()` leaked until the thread exit. Their stacks also leaked except when `fiber_set_joinable(..., true)` was used.
- Fixed a crash when Tarantool was launched with multiple `-e` or `-l` options without a space between the option and the value (gh-5747).

### Vinyl

- Immediate removal of compacted run files created after the last checkpoint optimization now works for replica's initial JOIN stage (gh-6568).
- Fixed crash during recovery of a secondary index in case the primary index contains incompatible phantom tuples (gh-6778).

### Replication

- Fixed replicas failing to bootstrap when master is just re-started (gh-6966).

### LuaJIT

- Fixed top part of Lua stack (red zone, free slots, top slot) unwinding in `lj-stack` command.
- Added the value of `g->gc.mmudata` field to `lj-gc` output.
- `string.char()` builtin recording is fixed in case when no arguments are given (gh-6371, gh-6548).
- Actually made JIT respect `maxirconst` trace limit while recording (gh-6548).

### Box

- Added iterator type checking and allow passing iterator as a `box.index.{ALL,GT,...}` directly (gh-6501).

### Recovery

- When `force_recovery` cfg option is set, Tarantool is able to boot from `snap`/`xlog` combinations where `xlog` covers changes committed both before and after `snap` creation. For example, `0...0.xlog`, covering everything up to `vclock {1: 15}` and `0...09.snap`, corresponding to `vclock {1: 9}` (gh-6794).

## Tarantool 1.10.12

Released on 2021-12-22.

- Release: v. 1.10.12.
- Tag: `1.10.12`.

### Overview

1.10.12 is the next stable release in the *long-term support (LTS) version* 1.10.x release series.

The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release introduces 3 improvements and resolves roughly 10 issues since the 1.10.11 version.

### Compatibility

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 1.10.x series.

### Functionality added or changed

#### Build

- Stop support of Ubuntu Trusty (14.04). (gh-6502)
- Bump debian package compatibility level to 10 (gh-5429). Bump minimal required debhelper to version 10 (except for Ubuntu Xenial).

### Bugs fixed

#### Core

- Fixed a crash caused by a race between box.session.push() and closing connection (gh-6520).
- Fixed crash in case a fiber changing box.cfg.listen is woken up (gh-6480).
- Fixed box.cfg.listen not reverted to the old address in case the new one is invalid (gh-6092).

#### Replication

- Fixed replica reconnecting to a living master on any `box.cfg{replication=...}` change. Such reconnects could lead to replica failing to restore connection for `replication_timeout` seconds (gh-4669).

#### LuaJIT

- Fixed the order VM registers are allocated by LuaJIT frontend in case of `BC_ISGE` and `BC_ISGT` (gh-6227).
- Fixed inconsistency while searching for an error function when unwinding a C protected frame to handle a runtime error (e.g. an error in __gc handler).

#### Lua

- When error is raised during encoding call results, auxiliary lightuserdata value is not removed from the main Lua coroutine stack. Prior to the fix it leads to undefined behaviour during the next usage of this Lua coroutine (gh-4617).
- Fixed Lua C API misuse, when the error is raised during call results encoding on unprotected coroutine and expected to be catched on the different one, that is protected (gh-6248).

#### Triggers

- Fixed possibility crash in case when trigger removes itself. Fixed possibility crash in case when someone destroy trigger, when it's yield (gh-6266).

#### Build

- The Debian package does not depend on binutils anymore (gh-6699).
- Fix build errors with glibc-2.34 (gh-6686).

### Tarantool 1.10.11

Released on 2021-08-19.

- Release v. 1.10.11.
- Tagged `1.10.11-0-gf0b0e7e`

#### Overview

1.10.11 is the next stable release in the *long-term support (LTS) version* 1.10.x release series.

The label «stable» means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release introduces 2 improvements and resolves roughly 18 issues since version *1.10.10*.

#### Compatibility

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 1.10.x series.

Some changes are labeled as **[Breaking change]**. It means that the old behavior was considered error-prone and therefore changed to protect users from unintended mistakes. However, there is a small probability that someone can rely on the old behavior, and this label is to bring attention to the things that have been changed.

#### Functionality added or changed

#### LuaJIT

- Introduced support for `LJ_DUALNUM` mode in `luajit-gdb.py` (gh-6224).

### Build

- Fedora 34 builds are now supported (gh-6074).
- Fedora 28 and 29 builds are no longer supported.

### Bugs fixed

### Core

- **[Breaking change]** `fiber.wakeup()` in Lua and `fiber_wakeup()` in C became NOP on the currently running fiber. Previously they allowed "ignoring" the next yield or sleep, which resulted in unexpected erroneous wake-ups. Calling these functions right before `fiber.create()` in Lua or `fiber_start()` in C could lead to a crash (in debug build) or undefined behaviour (in release build) (gh-6043).

  There was a single use case for the previous behaviour: rescheduling in the same event loop iteration, which is not the same as `fiber.sleep(0)` in Lua and `fiber_sleep(0)` in C. It could be done in the following way:

  in C:

  ```
  fiber_wakeup(fiber_self());
  fiber_yield();
  ```

  and in Lua:

  ```
  fiber.self():wakeup()
  fiber.yield()
  ```

  To get the same effect in C, one can now use `fiber_reschedule()`. In Lua, it is now impossible to reschedule the current fiber directly in the same event loop iteration. One can reschedule self through a second fiber, but it is strongly discouraged:

  ```
  -- do not use this code
  local self = fiber.self()
  fiber.new(function() self:wakeup() end)
  fiber.sleep(0)
  ```

- Fixed memory leak on `box.on_commit()` and `box.on_rollback()` (gh-6025).
- Fixed invalid results of the `json` module's `encode` function when it was used from the Lua garbage collector. For example, this could happen in functions used as `ffi.gc()` (gh-6050).
- Fixed console client connection failure in case of request timeout (gh-6249).
- Added a missing broadcast to `net.box.future:discard()` so that now fibers waiting for a request result wake up when the request is discarded (gh-6250).
- Fixed a bug when iterators became invalid (up to crash) after schema change (gh-6147).
- Fixed crash in case of reloading a compiled module when the new module lacks some functions present in the former code. In turn, this event triggers a fallback procedure where we restore old functions, but instead of restoring each function, we process a sole entry only, leading to the crash later when these restored functions are called (gh-5968).

#### Vinyl

- Fixed possible keys divergence during secondary index build, which might lead to missing tuples (gh-6045).
- Fix crash which may occur while switching read_only mode due to duplicating transaction in tx writer list (gh-5934).
- Fixed the race between Vinyl garbage collection and compaction that resulted in a broken vylog and recovery failure (gh-5436).

#### Replication

- Fixed replication occasionally stopping with `ER_INVALID_MSGPACK` when the replica is under high load (gh-4040).

#### LuaJIT

- Fixed optimization for single-char strings in the `IR_BUFPUT` assembly routine.
- Fixed slots alignment in the `lj-stack` command output when `LJ_GC64` is enabled (gh-5876).
- Fixed dummy frame unwinding in the `lj-stack` command.
- Fixed detection of inconsistent renames even in the presence of sunk values (gh-4252, gh-5049, gh-5118).
- Fixed the VM register allocation order provided by LuaJIT frontend in case of `BC_ISGE` and `BC_ISGT` (gh-6227).

#### Lua

- When an error occurs during encoding call results, the auxiliary lightuserdata value is not removed from the main Lua coroutine stack. Before the fix, it led to undefined behaviour during the next usage of this Lua coroutine (gh-4617).
- Fixed a Lua C API misuse when the error is raised during call results encoding in an unprotected coroutine and expected to be caught in a different, protected coroutine (gh-6248).

#### Triggers

- Fixed a possible crash in case trigger removes itself. Fixed a possible crash in case someone destroys a trigger when it yields (gh-6266).

### Tarantool 1.10.10

Released on 2021-04-21.

- Release v. 1.10.10.
- Tagged `1.10.10-0-gaea7ae7`

#### Общие сведения

1.10.10 is the next stable release in the *long-term support (LTS) version* 1.10.x release series.

The label «stable» means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release introduces 12 improvements and resolves roughly 3 issues since version *1.10.9*.

#### Совместимость

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol.

Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 1.10.x series.

#### Изменения или добавления функциональности

#### Build

- Tarantool build infrastructure now requires CMake version 3.1 or later.
- Binary packages for Fedora 33 (gh-5502) are now available.
- Binary packages for CentOS 6 and Debian Jessie won't be published since this version.
- Backported the `-DENABLE_LTO=ON/OFF` CMake option (gh-3117, gh-3743). It is useful for building packages for Fedora 33 (gh-5502).
- The built-in zstd is upgraded from v1.3.3 to v1.4.8 (part of gh-5502).
- `libcurl` symbols in the case of bundled libcurl are now exported (gh-5223, gh-5542).
- SMTP and SMTPS protocols are now enabled in the bundled `libcurl` (gh-4559).
- The `libcurl` headers are now shipped to system path `${PREFIX}/include/tarantool` when `libcurl` is included as a bundled library or in a static build (gh-4559).

#### Тестирование

- Tarantool CI/CD has migrated to GitHub Actions (gh-5662).
- Implemented a self-sufficient LuaJIT testing environment. As a result, LuaJIT build system is now partially ported to CMake and all testing machinery is enclosed within the tarantool/luajit repository (gh-4862, gh-5470).
- Python 3 is now the default in the test infrastructure (gh-5652).

#### Исправленные ошибки

#### Core

- Extensive usage of `uri` and `uuid` modules with debug log level no longer leads to crashes or corrupted results of the functions from these modules. Same problem is resolved for using these modules from the callbacks passed to `ffi.gc()`, and for some functions from the modules `fio`, `box.tuple`, and `iconv` (gh-5632).

#### Lua

- Fixed the `-e` option, when `tarantool` used to enter the interactive mode when stdin is a TTY. Now, `tarantool -e 'print"Hello"'` doesn't enter the interactive mode, but just prints «Hello» and exits (gh-5040).

#### Build

- Recovering with `force_recovery` option now deletes vylog files newer than the snapshot. It helps an instance recover after incidents during a checkpoint (gh-5823).

### Tarantool 1.10.9

Release: v. 1.10.9

Date: 2020-12-30 Tag: 1.10.9-0-g720ffdd23

#### Общие сведения

1.10.9 is the next stable release in the *long-term support (LTS) version* 1.10.x release series. The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release introduces one improvement and resolves roughly 7 issues since the 1.10.8 version.

### Совместимость

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol. Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 1.10.x series.

### Изменения или добавления функциональности

#### Build

- Deploy packages for Fedora 32 (gh-4966).
- Deploy packages for Debian Bullseye (gh-5638).

#### Разное

- Don't start an 'example' instance after installing Tarantool (gh-4507).

  Before this release the `tarantool` package for Debian and Ubuntu used to automatically enable and start an „example" instance, which would listen on the TCP port `3301`. Since this release the instance file is installed to `/etc/tarantool/instances.available/example.lua`, but isn't enabled by default and not started anymore. You can enable and start it with the following commands:

```
ln -s /etc/tarantool/instances.available/example.lua \
    /etc/tarantool/instances.enabled/example.lua
systemctl start tarantool@example
```

  Existing configuration will not be updated automatically at package update. If you don't the need example instance, you can stop and disable it with the following commands:

```
systemctl stop tarantool@example
rm /etc/tarantool/instances.enabled/example.lua
```

### Исправленные ошибки

#### Core

- `fiber.cond:wait()` now correctly throws an error when a fiber is cancelled, instead of ignoring the timeout and returning without any signs of an error (gh-5013).
- Fixed a memory corruption issue, which was most visible on macOS, but could affect any system (gh-5312).
- A dynamic module now gets correctly unloaded from memory in case of an attempt to load a non-existing function from it (gh-5475).
- The fiber region (the box region) won't be invalidated on a read-only transaction (gh-5427, gh-5623).

#### LuaJIT

- Dispatching `__call` metamethod no longer causes address clashing (gh-4518, (gh-4649).
- Fixed a false positive panic when yielding in debug hook (gh-5649).

#### Lua

- An attempt to use a `net.box` connection which is not established yet now results in a correctly reported error (gh-4787).
- Fixed a hang which occurred when `tarantool` ran a user script with the `-e` option and this script exited with an error (like with `tarantool -e 'assert(false)'`) (gh-4983).

### Tarantool 1.10.8

Release: v. 1.10.8

Date: 2020-10-22 Tag: 1.10.8-1-ge69e130

#### Общие сведения

1.10.8 is the next stable release of the 1.10.x series. The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release resolves roughly 7 issues since the 1.10.7 version. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

#### Совместимость

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol. Please upgrade using the box.schema.upgrade() procedure to unlock all the new features of the 1.10.x series.

#### Изменения или добавления функциональности

#### Module API

- Exposed the box region, key_def and several other functions in order to implement external tuple.keydef and tuple.merger modules on top of them (gh-5273, gh-5384).

#### Исправленные ошибки

#### Core

- Fixed fibers switch-over to prevent JIT machinery misbehavior. Trace recording is aborted when fiber yields the execution. The yield occurring while the compiled code is being run (it's likely a function with a yield underneath called via LuaJIT FFI) leads to the platform panic (gh-1700, gh-4491).
- Fixed fibers switch-over to prevent implicit GC disabling. The yield occurring while user-defined __gc metamethod is running leads to the platform panic.

#### Lua

- Fixed unhandled Lua error that might lead to memory leaks and inconsistencies in `<space_object>:frommap()`, `<key_def_object>:compare()`, `<merge_source>:select()` (gh-5382).

#### LuaJIT

- Fixed the error occurring on loading luajit-gdb.py with Python2 (gh-4828).

#### Разное

- Fixed potential lag on boot up procedure when system's password database is slow in access (gh-5034).

#### Module API

- Get rid of typedef redefinitions for compatibility with C99 (gh-5313).

### Tarantool 1.10.7

Release: v. 1.10.7

Date: 2020-07-17 Tag: 1.10.7-1-gb93a33a

#### Общие сведения

1.10.7 is the next stable release of the 1.10.x series. The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release resolves roughly 14 issues since 1.10.6. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

#### Совместимость

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol. Please upgrade using the box.schema.upgrade() procedure to unlock all the new features of the 1.10.x series.

#### Исправленные ошибки

##### Core

- Fixed a bug in C module reloading (gh-4945).
- Fixed races and corner cases in box (re)configuration (gh-4231).
- Fixed check of index field map size which led to crash (gh-5084).
- Fixed wrong mpsgpack extension type in an error message at decoding (gh-5017).

##### Lua

- Fixed error while closing socket.tcp_server socket (gh-4087).

##### LuaRocks

- Don't ruin rock name when freshly installing *.all.rock with dependencies (gh-4704).

##### Vinyl

- Fixed crash during compaction due to tuples with size exceeding vinyl_max_tuple_size setting (gh-4864).
- Fixed crash during recovery of vinyl index due to the lack of file descriptors (gh-4805).
- Fixed crash during executing upsert changing primary key in debug mode (gh-5005).
- Fixed crash due to triggered dump process during secondary index creation (gh-5042).
- Fixed crash/deadlock (depending on build type) during dump process scheduling and concurrent DDL operation (gh-4821).
- Fixed crash during read of prepared but still not yet not committed statement (gh-3395).
- Fixed squashing set and arithmetic upsert operations (gh-5106).
- Create missing folders for vinyl spaces and indexes if needed to avoid confusing fails of tarantool started from backup (gh-5090).
- Fixed crash during squash of many (more than 4000) upserts modifying the same key (gh-4957).

### Tarantool 1.10.6

Release: v. 1.10.6

Date: 2020-04-20 Tag: 1.10.6-1-g47c009a

#### Общие сведения

1.10.6 is the next stable release of the 1.10.x series. The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release resolves roughly 20 issues since 1.10.5. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

#### Совместимость

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol. Please upgrade using the box.schema.upgrade() procedure to unlock all the new features of the 1.10.x series.

#### Исправленные ошибки

#### Core

- `fiber.storage` is cleaned between requests, and can be used as a request-local storage. Previously `fiber.storage` could contain some old values in the beginning of an iproto request execution, and it needed to be nullified manually. Now the cleanup is unneeded (gh-4662).
- `tuple`/`space`/`index`:`update()`/`upsert()` were fixed not to turn a value into an infinity when a float value was added to or subtracted from a float column and exceeded the float value range (gh-4701).
- Make RTREE indexes handle the out of memory error: before this fix, OOM during the recovery of an RTREE index resulted in segmentation fault (gh-4619).
- Add cancellation guard to avoid WAL thread stuck (gh-4127).

#### Репликация

- Fix the rebootstrap procedure not working if the replica itself is listed in `box.cfg.replication` (gh-4759).
- Fix possible user password leaking via replication logs (gh-4493).
- Local space operations are now counted in 0th vclock component. Every instance may have its own 0-th vclock component not matching others'. Local space operations are not replicated at all, even as NOPs (gh-4114).
- Gc consumers are now ordered by their vclocks and not by vclock signatures. Only the WALS that contain no entries needed by any of the consumers are deleted (gh-4114).

### Lua

- json: `:decode()` doesn't spoil instance's options with per-call ones (when it is called with the second argument) (gh-4761).
- `os.environ()` is now changed when `os.setenv()` is called (gh-4733).
- `netbox.self:call/eval()` now returns the same types as `netbox_connection:call/eval`. Previously it could return a tuple or `box.error` cdata (gh-4513).
- libev: backport fix for listening for more then 1024 file descriptors on Mac OS (gh-3867).

### HTTP client

- When building Tarantool with bundled `libcurl`, link it with the `c-ares` library by default (gh-4591).

### LuaJIT

- `__pairs`/`__ipairs` metamethods handling is removed since we faced the issues with the backward compatibility between Lua 5.1 and Lua 5.2 within Tarantool modules as well as other third party code (gh-4770).
- Introduce `luajit-gdb.py` extension with commands for inspecting LuaJIT internals. The extension obliges one to provide gdbinfo for `libluajit`, otherwise loading fails. The extension provides the following commands:
  - `lj-arch` dumps values of LJ_64 and LJ_GC64 macro definitions
  - `lj-tv` dumps the type and GCobj info related to the given TValue
  - `lj-str` dumps the contents of the given GCstr
  - `lj-tab` dumps the contents of the given GCtab
  - `lj-stack` dumps Lua stack of the given lua_State
  - `lj-state` shows current VM, GC and JIT states
  - `lj-gc` shows current GC stats
- Fix string to number conversion: current implementation respects the buffer length (gh-4773).
- **"FFI sandwich"(*) detection is introduced. If sandwich is detected** while trace recording the recording is aborted. The sandwich detected while mcode execution leads to the platform panic.
- luaJIT_setmode call is prohibited while mcode execution and leads to the platform panic.

(*) The following stack mix is called FFI sandwich:

Lua-FFI -> C routine -> Lua-C API -> Lua VM

This sort of re-entrancy is explicitly not supported by LuaJIT compiler. For more info see (gh-4427).

#### Vinyl

- Fix assertion fault due to triggered dump process during secondary index build (gh-4810).

#### Разное

- Fix crashes at attempts to use `-e` and `-l` command line options concatenated with their values, like this: `-eprint(100)` (gh-4775).
- Update `libopenssl` version to 1.1.1f since the previous one was EOLed (gh-4830).

#### Building from sources

- Fix static build (`-DBUILD_STATIC=ON`) when `libunwind` depends on `liblzma` (gh-4551).

### Tarantool 1.10.5

Release: v. 1.10.5

Date: 2020-01-14 Tag: 1.10.5-0-g83a2ae9

#### Общие сведения

1.10.5 is the next stable release of the 1.10.x series. The label "stable" means there are 1.10.x-based applications running in production for quite a while without known crashes, incorrect results or other showstopper bugs.

This release resolves roughly 30 issues since 1.10.4. There may be bugs in less common areas. If you find any, feel free to report an issue at GitHub.

#### Совместимость

Tarantool 1.10.x is backward compatible with Tarantool 1.9.x in binary data layout, client-server protocol and replication protocol. Please upgrade using the `box.schema.upgrade()` procedure to unlock all the new features of the 1.10.x series.

#### Изменения или добавления функциональности

#### Lua

- Exit gracefully when a main script throws an error: notify systemd, log the error (gh-4382).
- Enable `__pairs` and `__ipairs` metamethods from Lua 5.2 (gh-4560). We still conform Lua 5.1 API that is not always compatible with Lua 5.2. The change is only about those metamethods.

### Разное

- Add package builds and deployment for the following Linux distros:
- Ubuntu 19.10 Eoan Ermine, (gh-4583)
- CentOS 8, (gh-4543)
- Fedora 31, (gh-4612)

### Исправленные ошибки

#### Core

- fiber: make sure the guard page is created; refuse to create a new fiber otherwise (gh-4541). It is possible in case of heavy memory usage, say, when there is no resources to split VMAs.
- Forbid `00000000-0000-0000-0000-000000000000` as the value of `box.cfg({<...>})` options: `replicaset_uuid` and `instance_uuid` (gh-4282). It did not work as expected: the nil UUID was treated as absence of the value.
- Update cache of universe privileges without reconnect (gh-2763).
- net.box: don't fire the `on_connect` trigger on schema update (gh-4593). Also don't fire the `on_disconnect` trigger if a connection never entered into the `active` state (e.g. when the first schema fetch is failed).
- func: fix use-after-free on function unload. fce9cf96
- Don't destroy a session until `box.session.on_disconnect(<...>)` triggers are finished (gh-4627). This means, for example, that `box.session.id()` can be safely invoked from the `on_disconnect` trigger. Before this change `box.session.id()` returned garbage (usually 0) after yield in the `on_disconnect` trigger. *Note*: tarantool/queue module is affected by this problem in some scenarios. It is especially suggested to update Tarantool at least to this release if you're using this module.
- Handle OOM gracefully during allocating a buffer for binary protocol response. 5c5a4e2d
- func: Fix `box.schema.func.drop(<..>)` to unload unused modules (gh-4648). Also fix `box.schema.func.create(<..>)` to avoid loading a module again when another function from the module is loaded.
- Encode Lua number -2^63 as integer in `msgpack.encode()` and box's functions (gh-4672).

#### Репликация

- Prefer to bootstrap a replica from a fully bootstrapped instance rather than from an instance that is in the process of bootstrapping (gh-4527).

  This change enables the case when two nodes (B, C) are being bootstrapped simultaneously using the one that is already bootstrapped (A), while A is configured to replicate from {B, C} and B – from {A, C}.
- Return immediately from `box.cfg{<...>}` when an instance is reconfigured with `replication_connect_quorum = 0` (gh-3760).

  This change also fixes the behaviour of reconfiguration with non-zero `replication_connect_quorum`: `box.cfg{<...>}` returns immediately regardless of whether connections to upstreams are established.
- Auto reconnect a replica if password is invalid (gh-4550).

- Use empty password when a URI in `box.cfg{replication = <...>}` is like `login@host:port` (gh-4605).

  This behaviour matches the net.box's one now. Explicit `login:@host:port` was necessary before, otherwise a replica displayed the following error:

  > Missing mandatory field „tuple“ in request

- Fix segfault during replication configuration (`box.cfg{replication = <...>}` call) (gh-4440, gh-4576, gh-4586, gh-4643).
- Apply replication settings of `box.cfg({<...>})` in the strict order (gh-4433).

### Lua

- Fix handling of a socket read error in the console client (`console.connect(<URI>)` or `tarantoolctl connect/enter <...>`). 1f86e6cc
- Handle the "not enough memory" error gracefully when it is raised from `lua_newthread()` (gh-4556). There are several cases when a new Lua thread is created:
  - Start executing a Lua function call or an eval request (from a binary protocol, SQL or with `box.func.<...>:call()`).
  - Create of a new fiber.
  - Start execution of a trigger.
  - Start of encoding into a YAML format (`yaml.encode()`).
- Fix stack-use-after-scope in `json.decode()` (gh-4637).
- Allow to use `cdata<struct ibuf *>` (e.g. buffer.IBUF_SHARED) as the argument to `msgpack.encode()`. 6d38f0c5 Before this change the `cdata<struct ibuf>` type was allowed, but not the corresponding pointer type.
- A pointer returned by `msgpack.decode*(cdata<[char] const *>)` functions can be assigned to buffer.rpos now (and the same for msgpackffi) (gh-3926).

  All those functions now return `cdata<char *>` or `cdata<const char *>` depending of a passed argument. Example of the code that did not work: `res, buf.rpos = msgpack.decode(buf.rpos, buf:size())`.
- Fix race in `fio.mktree()` when two tarantool processes create the same directory tree simultaneously (gh-4660). This problem affects tarantool/cartrige, see cartrige#gh-382.

### HTTP client

- Disable verbose mode when `{verbose = false}` is passed. 28f8a5eb
- Fix assertion fail after a curl write error (gh-4232).

### LuaJIT

- Fix the "Data segment size exceeds process limit" error on FreeBSD/x64: do not change resource limits when it is not necessary (gh-4537).
- fold: keep type of emitted CONV in sync with its mode. LuaJIT#524 This fixes the following assertion fail:

  asm_conv: Assertion `((IRType)((ir->t).irt & IRT_TYPE)) != st' failed

### Разное

- Fix CLI boolean options handling in `tarantoolctl cat <...>`, such as `--show-system` (gh-4076).
- Fix segfault (out of bounds access) when a stack unwinding error occurs at backtrace printing (gh-4636). Backtrace is printed on SIGFPE and SIGSEGV signals or when LuaJIT finds itself in the unrecoverable state (`lua_atpanic()`).
- Clear terminal state on panic (gh-4466).

### Building from sources

- Fix OpenSSL linking problems on FreeBSD (gh-4490).
- Fix linking problems on Mac OS when several toolchains are in PATH (gh-4587).
- Fix GCC 9 warning on `strncpy()` (gh-4515).

### Tarantool 1.10

Release: v. 1.10.0

**Версия 1.10.4**

Тип версии: стабильная (lts). Дата выхода: 2019-09-26. Тег: 1-10-4.

Release: v. 1.10.4.

Общие сведения

1.10.4 представляет собой очередную *стабильную (lts)* версию в серии 1.10. Пометка «стабильная» означает, что некоторые системы в течение определенного времени успешно отработали в производственной среде без известных сбоев, ненадежных результатов и прочих неисправностей.

Данная версия содержит около 50 исправлений по сравнению с версией 1.10.3.

Совместимость

Tarantool 1.10.x обратно совместим с Tarantool 1.9.x в том, что касается структуры бинарных данных, клиент-серверного протокола и протокола репликации. Чтобы воспользоваться новыми функциями серии 1.10.x, *обновите* версию 1.9 с помощью процедуры `box.schema.upgrade()`.

Изменения или добавления функциональности

- **(Engines) Improve dump start/stop logging. When initiating memory dump, print** how much memory is going to be dumped, expected dump rate, ETA, and the recent write rate. Upon dump completion, print observed dump rate in addition to dump size and duration.

- **(Engines) Look up key in reader thread. If a key isn't found in the tuple cache,** we fetch it from a run file. In this case disk read and page decompression is done by a reader thread, however key lookup in the fetched page is still performed by the TX thread. Since pages are immutable, this could as well be done by the reader thread, which would allow us to save some precious CPU cycles for TX (gh-4257).
- **(Core) Improve *box.stat.net*** (gh-4150).
- **(Core) Add `idle` to downstream status in `box.info`.** When a relay sends a row it updates `last_row_time` value with the current time. When `box.info()` is called, `idle` is set to `current_time - last_row_time`.
- **(Replication) Print corrupted rows on decoding error.** Improve row printing to log. Print the header row by row, 16 bytes in a row, and format output to match `xxd` output:

```
[001] 2019-04-05 18:22:46.679 [11859] iproto V> Got a corrupted row:
[001] 2019-04-05 18:22:46.679 [11859] iproto V> 00000000: A3 02 D6 5A E4 D9 E7␣
↪68 A1 53 8D 53 60 5F 20 3F
[001] 2019-04-05 18:22:46.679 [11859] iproto V> 00000010: D8 E2 D6 E2 A3 02 D6␣
↪5A E4 D9 E7 68 A1 53 8D 53
```

- **(Lua) Add type of operation to space *trigger parameters*.** For example, a trigger function may now look like this:

```
function before_replace_trig(old, new, space_name, op_type)
    if op_type == 'INSERT' then
        return old
    else
        return new
    end
end
```

  (gh-4099).
- **(Lua) Add `debug.sourcefile()` and `debug.sourcedir()` helpers** (and `debug.__file__` and `debug.__dir__` shortcuts) to determine the location of a current Lua source file. Part of (gh-4193).
- **(HTTP client) Add `max_total_connections` option in addition to `max_connections`** to allow more fine-grained tuning of `libcurl` connection cache. Don't restrict the total connections` with a constant value by default, but use `libcurl`'s default, which scales the threshold according to easy handles count (gh-3945).

Исправленные ошибки

- **(Vinyl) Fix assertion failure in `vy_tx_handle_deferred_delete`** (gh-4294).
- (Vinyl) Don't purge deleted runs from vylog on compaction. Cherry-picked from (gh-4218).
- **(Vinyl) Don't throttle DDL** (gh-4238).
- **(Vinyl) Fix deferred DELETE statement lost on commit** (gh-4248).
- **(Vinyl) Fix assertion while recovering dumped statement** (gh-4222).
- **(Vinyl) Reset dump watermark after updating memory limit** (gh-3864).
- **(Vinyl) Be pessimistic about write rate when setting dump watermark** (gh-4166).
- **(Vinyl) Fix crash if space is dropped while space.get is reading from it** (gh-4109).
- **(Vinyl) Fix crash during index build** (gh-4152).

- **(Vinyl) Don't compress L1 runs** (gh-2389).
- (Vinyl) Считать операторы, пропущенные при чтении.
- (Vinyl) Принять во внимание поиск первичного ключа при учете задержек.
- (Vinyl) Исправить зависание `vy_range_update_compaction_priority`.
- (Vinyl) Освобождать область при фиксации vylog вместо сброса и очищать после выделения заместителя оператора.
- (Vinyl) Еще увеличить ограничение на количество открытых файлов в файле `systemd`.
- (Vinyl) Увеличить минимальный размер диапазона до 128 Мбайт
- **(Memtx) Cancel checkpoint thread at exit** (gh-4170).
- **(Core) Fix crash for update with empty tuple** (gh-4041).
- **(Core) Fix use-after-free in `space_truncate`** (gh-4093).
- **(Core) Fix error while altering index with sequence** (gh-4214).
- **(Core) Detect a new invalid json path case** (gh-4419).
- **(Core) Fix empty password authentication** (gh-4327).
- (Ядро) Исправить размер массива `txn::sub_stmt_begin`.
- (Ядро) Учитывать `index.pairs` в `box.stat.SELECT()`.
- **(Replication) Disallow bootstrap of read-only masters** (gh-4321).
- **(Replication) Enter orphan mode on manual replication configuration change** (gh-4424).
- (Репликация) Задать значение `now` для `last_row_time` в `relay_new` и `relay_start`. Проблема 4431.
- **(Replication) Stop relay on subscribe error** (gh-4399).
- **(Replication) Init `coio` watcher before join/subscribe** (gh-4110).
- **(Replication) Allow to change instance id during join** (gh-4107).
- (Репликация) Исправить логику сборки мусора.
- (Репликация) Вернуть проверку границ пакета для iproto.
- (Репликация) Не прерывать репликацию при ER_UNKNOWN_REPLICA.
- (Репликация) Уменьшить воздействие фрагментации буфера ввода при большом `cfg.readahead`.
- (Репликация) Исправить обновление с 1.7 (не распознает тип запроса IPROTO_VOTE).
- **(Replication) Fix memory leak in call / eval in the case when a transaction** is not committed (gh-4388).
- **(Lua) Fix `fio.mktree()` error reporting** (gh-4044).
- **(Lua) Fix segfault on `ffi.C_say()` without filename** (gh-4336).
- **(Lua) Fix segfault on `json.encode()` on a recursive table** (gh-4366).
- **(Lua) Fix `pwd.getpwall()` and `pwd.getgrall()` hang on CentOS 6** and FreeBSD 12 (gh-4447, gh-4428).
- **(Lua) Fix a segfault during initialization of a cipher from `crypto` module** (gh-4223).
- **(HTTP client) Reduce stack consumption during waiting for a DNS resolving result** (gh-4179).

- **(HTTP client) Increase max outgoing header size to 8 KiB** (gh-3959).
- **(HTTP client) Verify «headers» option stronger** (gh-4281, gh-3679).
- **(HTTP client) Use bundled `libcurl` rather than system-wide by default** (gh-4318, gh-4180, gh-4288, gh-4389, gh-4397).
- **(HTTP client) This closes several known problems that were fixed in recent `libcurl`** versions, including segfaults, hangs, memory leaks and performance problems.
- **(LuaJIT) Fix overflow of snapshot map offset** Part of (gh-4171).
- **(LuaJIT) Fix rechaining of pseudo-resurrected string keys** Part of (gh-4171).
- **(LuaJIT) Fix fold machinery misbehaves** (gh-4376).
- **(LuaJIT) Fix for `debug.getinfo(1,'>S')`** (gh-3833).
- **(LuaJIT) Fix `string.find` recording** (gh-4476).
- (LuaJIT) Исправлена ошибка нетонущих 64-битных указателей.
- (Разное) Еще увеличить предел количества открытых файлов в файле `systemd`.
- **(Misc) Raise error in `tarantoolctl` when `box.cfg()` isn't called** (gh-3953).
- **(Misc) Support `systemd`'s NOTIFY_SOCKET on OS X** (gh-4436).
- **(Misc) Fix `coio_getaddrinfo()` when 0 timeout is passed** (affects `netbox`'s `connect_timeout`) (gh-4209).
- **(Misc) Fix `coio_do_copyfile()` to perform truncate of destination** (affects `fio.copyfile()`) (gh-4181).
- (Разное) Сделать подсказки в `coio_getaddrinfo()` необязательными.
- **(Misc) Validate `msgpack.decode()` cdata size argument** (gh-4224).
- **(Misc) Fix linking with static `openssl` library** (gh-4437).

Устаревшие функции

- **(Core) Deprecate `rows_per_wal` in favor of `wal_max_size`.** Part of (gh-3762).

### Версия 1.10.3

Тип версии: стабильная (lts). Дата выхода: 2019-04-01. Тег: 1-10-3.

Release: v. 1.10.3.

Общие сведения

1.10.3 представляет собой очередную *стабильную (lts)* версию в серии 1.10. Пометка «стабильная» означает, что некоторые системы в течение определенного времени успешно отработали в производственной среде без известных сбоев, ненадежных результатов и прочих неисправностей.

Данная версия содержит 69 исправлений по сравнению с версией 1.10.2.

Совместимость

Tarantool 1.10.x обратно совместим с Tarantool 1.9.x в том, что касается структуры бинарных данных, клиент-серверного протокола и протокола репликации. Чтобы воспользоваться новыми функциями серии 1.10.x, *обновите* версию 1.9 с помощью процедуры `box.schema.upgrade()`.

Изменения или добавления функциональности

- **(Engines) Randomize vinyl index compaction** (gh-3944).
- **(Engines) Throttle tx thread if compaction doesn't keep up with dumps** (gh-3721).

- **(Engines) Do not apply run_count_per_level to the last level** (gh-3657).
- **(Server) Report the number of active iproto connections** (gh-3905).
- **(Replication) Never keep a dead replica around if running out of disk space** (gh-3397).
- **(Replication) Report join progress to the replica log** (gh-3165).
- **(Lua) Expose snapshot status in box.info.gc()** (gh-3935).
- **(Lua) Show names of Lua functions in backtraces in fiber.info()** (gh-3538).
- **(Lua) Check if transaction opened** (gh-3518).

Исправленные ошибки

- **(Engines) Tarantool crashes if DML races with DDL** (gh-3420).
- **(Engines) Recovery error if DDL is aborted** (gh-4066).
- **(Engines) Tarantool could commit in the read-only mode** (gh-4016).
- **(Engines) Vinyl iterator crashes if used throughout DDL** (gh-4000).
- **(Engines) Vinyl doesn't exit until dump/compaction is complete** (gh-3949).
- **(Engines) After re-creating secondary index no data is visible** (gh-3903).
- **(Engines) box.info.memory().tx underflow** (gh-3897).
- **(Engines) Vinyl stalls on intensive random insertion** (gh-3603).
- **(Server) Newer version of libcurl explodes fiber stack** (gh-3569).
- **(Server) SIGHUP crashes tarantool** (gh-4063).
- **(Server) checkpoint_daemon.lua:49: bad argument #2 to „format“** (gh-4030).
- **(Server) fiber:name() show only part of name** (gh-4011).
- **(Server) Second hot standby switch may fail** (gh-3967).
- **(Server) Updating box.cfg.readahead doesn't affect existing connections** (gh-3958).
- **(Server) fiber.join() blocks in „suspended“ if fiber has cancelled** (gh-3948).
- **(Server) Tarantool can be crashed by sending gibberish to a binary socket** (gh-3900).
- **(Server) Stored procedure to produce push-messages never breaks on client disconnect** (gh-3559).
- **(Server) Tarantool crashed in lj_vm_return** (gh-3840).
- **(Server) Fiber executing box.cfg() may process messages from iproto** (gh-3779).
- **(Server) Possible regression on nosqlbench** (gh-3747).
- **(Server) Assertion after improper index creation** (gh-3744).
- **(Server) Tarantool crashes on vshard startup (lj_gc_step)** (gh-3725).
- **(Server) Do not restart replication on box.cfg if the configuration didn't change** (gh-3711).
- **(Replication) Applier times out too fast when reading large tuples** (gh-4042).
- **(Replication) Vinyl replica join fails** (gh-3968).
- **(Replication) Error during replication** (gh-3910).

- **(Replication) Downstream status doesn't show up in replication.info unless the channel is broken** (gh-3904).
- **(Replication) replication fails: tx checksum mismatch** (gh-3883).
- **(Replication) Rebootstrap crashes if master has replica's rows** (gh-3740).
- **(Replication) After restart tuples revert back to their old state which was before replica sync** (gh-3722).
- **(Replication) Add vclock for safer hot standby switch** (gh-3002).
- **(Replication) Master row is skipped forever in case of wal write failure** (gh-2283).
- **(Lua) space:frommap():tomap() conversion fail** (gh-4045).
- **(Lua) Non-informative message when trying to read a negative count of bytes from socket** (gh-3979).
- **(Lua) space:frommap raise «tuple field does not match...» even for nullable field** (gh-3883).
- **(Lua) Tarantool crashes on net.box.call after some uptime with vshard internal fiber** (gh-3751).
- **(Lua) Heap use after free in lbox_error** (gh-1955).
- **(Misc) http.client doesn't honour „connection: keep-alive“** (gh-3955).
- **(Misc) net.box wait_connected is broken** (gh-3856).
- **(Misc) Mac build fails on Mojave** (gh-3797).
- **(Misc) FreeBSD build error: no SSL support** (gh-3750).
- **(Misc) „http.client“ sets invalid (?) reason** (gh-3681).
- **(Misc) Http client silently modifies headers when value is not a «string» or a «number»** (gh-3679).
- **(Misc) yaml.encode uses multiline format for „false“ and „true“** (gh-3662).
- **(Misc) yaml.encode encodes „null“ incorrectly** (gh-3583).
- **(Misc) Error object message is empty** (gh-3604).
- **(Misc) Log can be flooded by warning messages** (gh-2218).

Устаревшие функции

- Опция `console=true` для *net.box.new()* объявлена устаревшей.

### Версия 1.10.2

Тип версии: стабильная (lts). Дата выхода: 2018-10-13. Тег: 1-10-2.

Release: v. 1.10.2.

Данная сборка представляет собой первую *стабильную (lts)* версию в серии 1.10. Кроме того, Tarantool 1.10.2 представляет собой мажорную версию, версия Tarantool 1.9.2 объявлена устаревшей. Это обновление содержит 95 исправлений по сравнению с версией 1.9.2.

Tarantool 1.10.x обратно совместим с Tarantool 1.9.x в том, что касается структуры бинарных данных, клиент-серверного протокола и протокола репликации. *Обновление* можно произвести с помощью процедуры `box.schema.upgrade()`.

Цель данного релиза – значительно повысить стабильность `vinyl'a` и реализовать автоматическую повторную настройку набора реплик в Tarantool.

Изменения или добавления функциональности:

- (Engines) support ALTER for non-empty vinyl spaces (gh-1653).
- (Engines) tuples stored in the vinyl cache are not shared among the indexes of the same space (gh-3478).
- (Engines) keep a stack of UPSERTS in `vy_read_iterator` (gh-1833).
- (Engines) `box.ctl.reset_stat()`, a function to reset vinyl statistics (gh-3198).
- (Server) *configurable syslog destination* (gh-3487).
- (Server) allow different nullability in indexes and format (gh-3430).
- (Server) allow to *back up any checkpoint*, not just the last one (gh-3410).
- (Server) a way to detect that a Tarantool process was started / restarted by `tarantoolctl` (*TARANTOOLCTL and TARANTOOL_RESTARTED* env vars) (gh-3384, gh-3215).
- (Server) *net_msg_max* configuration parameter to restrict the number of allocated fibers (gh-3320).
- (Replication) display the connection status if the downstream gets disconnected from the upstream (*box.info.replication.downstream.status* = `disconnected`) (gh-3365).
- (Replication) *replica-local spaces* (gh-3443)
- (Replication) *replication_skip_conflict*, a new option in `box.cfg{}` to skip conflicting rows in replication (gh-3270)
- (Replication) remove old snapshots which are not needed by replicas (gh-3444)
- (Replication) log records which tried to commit twice (gh-3105)
- (Lua) new function *fiber.join()* (gh-1397).
- (Lua) new option `names_only` to *tuple:tomap()* (gh-3280).
- (Lua) support custom rock servers (`server` and `only-server` options for *tarantoolctl rocks* command) (gh-2640).
- (Lua) expose `on_commit`/`on_rollback` triggers to Lua (gh-857).
- (Lua) new function *box.is_in_txn()* to check if a transaction is open (gh-3518).
- (Lua) tuple field access via a json path (by *number*, *name*, and *path*); (gh-1285).
- (Lua) new function *space:frommap()* (gh-3282).
- (Lua) new module *utf8* that implements libicu's bindings for use in Lua (gh-3290, gh-3385).

### Tarantool 1.9

Release: v. 1.9.0

**Версия 1.9.0**

Тип версии: стабильная. Дата выхода: 2018-02-26. Тег: 1.9.0-4-g195d446.

Release: v. 1.9.0.

Эта версия следует за стабильной версией 1.7.6. Цель данной версии – повысить стабилизацию vinyl'а и репликации типа мастер-мастер, для чего предусмотрено значительное количество новых функций. Следуйте инструкциям по загрузке по ссылке `https://tarantool.io/en/download/download.html` для установки пакета для вашей операционной системы.

Изменения или добавления функциональности:

- (Security) it is now possible to *block/unblock* users. (gh-2898).
- (Security) new function *box.session.euid()* to return effective user. Effective user can be different from authenticated user in case of `setuid` functions or `box.session.su`. (gh-2994).
- (Security) new *super* role, with superuser access. Grant „super“ to guest to disable access control. (gh-3022).
- (Security) *on_ auth* trigger is now fired in case of both successful and failed authentication. (gh-3039).
- (Replication/recovery) new replication configuration algorithm: if replication doesn't connect to replication_quorum peers in *replication_ connect_ timeout* seconds, the server start continues but the server enters the new *orphan* status, which is basically read-only, until the replicas connect to each other. (gh-3151 and gh-2958).
- (Репликация/восстановление) после включения репликации при запуске сервер не начинает обработку запросов на запись до *синхронизации* со всеми подключенными узлами.
- (Replication/recovery) it is now possible to explicitly set *instance_ uuid* and *replica set uuid* as configuration parameters. (gh-2967).
- (Replication/recovery) *box.once()* no longer fails on a read-only replica but waits. (gh-2537).
- (Replication/recovery) *force_ recovery* can now skip a corrupted xlog file. (gh-3076).
- (Replication/recovery) improved replication monitoring: *box.info.replication* shows peer ip:port and correct replication lag even for idle peers. (gh-2753 and gh-2689).
- (Application server) new *before* triggers which can be used for conflict resolution in master-master replication. (gh-2993).
- (Application server) *http client* now correctly parses cookies and supports http+unix:// paths. (gh-3040 and gh-2801).
- (Application server) `fio` rock now supports `file_exists()`, `rename()` works across filesystems, and `read()` without arguments reads the whole file. (gh-2924, gh-2751 and gh-2925).
- (Сервер приложений) ошибки в модуле `fio` соответствуют стандартам вызова функции в Tarantool и всегда возвращают сообщение об ошибке вместе с флагом ошибки.
- (Application server) `digest` rock now supports pbkdf2 password hashing algorithm, useful in PCI/DSS compliant applications. (gh-2874).
- (Application server) *box.info.memory()* provides a high-level overview of server memory usage, including networking, Lua, transaction and index memory. (gh-934).
- (Database) it is now possible to *add missing tuple fields* to an index, which is very useful when adding an index along with the evolution of the database schema. gh-2988).
- (Database) lots of improvements in field type support when creating or *altering* spaces and indexes. (gh-2893, gh-3011 and gh-3008).
- (Database) it is now possible to turn on *is_ nullable* property on a field even if the space is not empty, the change is instantaneous. (gh-2973).
- (Database) *logging* has been improved in many respects: individual messages (gh-1972, gh-2743, gh-2900), more logging in cases when it was useful (gh-3096, gh-2871).
- (Vinyl storage engine) it is now possible to make a *unique* vinyl index non-unique without index rebuild. (gh-2449).
- (Vinyl storage engine) improved UPDATE, REPLACE and recovery performance in presence of secondary keys. (gh-2289, gh-2875 and gh-3154).

- (Vinyl storage engine) *space:len()* and *space:bsize()* now work for vinyl (although they are still not exact). (gh-3056).
- (Vinyl storage engine) recovery speed has improved in presence of secondary keys. (gh-2099).
- (Builds) Alpine Linux support. (gh-3067).

### Tarantool 1.8

Release: v. 1.8.0

#### Версия 1.8.1

Release type: alpha. Release date: 2017-05-17. Tag: 1.8.1. Release: https://groups.google.com/forum/#!msg/tarantool-ru/XYaoqJpc544/mSvKrYwNAgAJ or v. 1.8.1.

This is an alpha release which delivers support for a substantial subset of the ISO/IEC 9075:2011 SQL standard, including joins, subqueries and views. SQL is a major feature of the 1.8 release series, in which we plan to add support for ODBC and JDBC connectors, SQL triggers, prepared statements, security and roles, and generally ensure SQL is a first class query language in Tarantool.

Изменения или добавления функциональности:

- A new function `box.sql.execute()` (later changed to `box.execute` in Tarantool 2.1) was added to query Tarantool databases using SQL statements, e.g.:

```
tarantool> box.sql.execute([[SELECT * FROM _schema]]);
```

- SQL and Lua are fully interoperable.
- New meta-commands introduced to Tarantool's console.

  You can now set input language to either SQL or Lua, e.g.:

```
tarantool> \set language sql
tarantool> SELECT * FROM _schema
tarantool> \set language lua
tarantool> print("Hello, world!")
```

- Most SQL statements are supported:
  - CREATE/DROP TABLE/INDEX/VIEW

```
tarantool> CREATE TABLE table1 (column1 INTEGER PRIMARY KEY, column2␣
↪VARCHAR(100));
```

  - INSERT/UPDATE/DELETE statements e.g.:

```
tarantool> INSERT INTO table1 VALUES (1, 'A');
...
tarantool> UPDATE table1 SET column2 = 'B';
```

  - **SELECT statements, including complex complicated variants which include** multiple JOINs, nested SELECTs etc. e.g.:

```
tarantool> SELECT sum(column1) FROM table1 WHERE column2 LIKE '_B' GROUP BY␣
↪column2;
```

  - WITH statements e.g.

```
tarantool> WITH cte AS ( SELECT SUBSTR(column2,1,2), column1 FROM table1␣
↪WHERE column1 >= 0) SELECT * FROM cte;
```

– SQL schema is persistent, so it is able to survive `snapshot()`/`restore()` sequence.

– SQL features are described in a *tutorial*.

### Tarantool 1.7

Release: v. 1.7.0

#### Версия 1.7.6

Release type: stable. Release date: 2017-11-07. Tag: 1.7.6-0-g7b2945d6c. Release: https://groups.google.com/forum/#!topic/tarantool/hzc7O2YDZUc or v. 1.7.6.

Данная сборка представляет собой очередную стабильную версию в серии 1.7. Это обновление содержит более 75 исправлений по сравнению с версией 1.7.5.

Что нового в Tarantool 1.7.6?

- В дополнение к *откату* транзакции, появился откат на определенную точку в пределах транзакции – поддержка *точки сохранения*.
- Появился новый объектный тип: *последовательности*. Устаревший вариант, *автоматическое увеличение*, объявлен устаревшим.
- В строковых индексах появилась *сортировка*.

Добавлены новые опции:

- *net_box* (время ожидания),
- функции *string*,
- *форматы* для спейса (имена и типы полей, задаваемые пользователем),
- *base64* (опция `urlsafe`), а также
- index *creation* (collation, *is-nullable*, field names).

Несовместимые изменения:

- Layout of `box.space._index` has been extended to support *is_nullable* and *collation* features. All new indexes created on columns with `is_nullable` or `collation` properties will have the new definition format. Please update your client libraries if you plan to use these new features (gh-2802).
- *fiber_name()* now raises an exception instead of truncating long fiber names. We found that some Lua modules such as expirationd use `fiber.name()` as a key to identify background tasks. If a name is truncated, this fact was silently missed. The new behavior allows to detect bugs caused by `fiber.name()` truncation. Please use `fiber.name(name, { truncate = true })` to emulate the old behavior (gh-2622).
- *space:format()* is now validated on DML operations. Previously `space:format()` was only used by client libraries, but starting from Tarantool 1.7.6, field types in `space:format()` are validated on the server side on every DML operation, and field names can be used in indexes and Lua code. If you used `space:format()` in a non-standard way, please update layout and type names according to the official documentation for space formats.

Изменения или добавления функциональности:

- Гибридная модель данных без схемы + со схемой. Раньше версии Tarantool позволяли хранить произвольный набор документов в формате MessagePack в спейсах. Начиная с версии Tarantool 1.7.6, можно использовать *space:format()* для определения условий и ограничений схемы для кортежей в спейсах. Определенные типы полей автоматически проверяются при каждой DML-операции, а определенные имена полей могут использоваться вместо номеров полей в Lua-коде. Добавлена новая функция *tuple:tomap()* для конвертации кортежа в Lua-словарь пар ключ-значение.
- Поддержка сортировки и Юникода. По умолчанию, когда Tarantool сопоставляет строки, он берет во внимание только числовое значение каждого байта в строке. Чтобы задействовать такое распределение, как в телефонных справочниках и словарях, в Tarantool версии 1.7.6 впервые поддерживается сортировка по Таблице сортировки символов Юникода по умолчанию (Default Unicode Collation Element Table (DUCET)) и в соответствии с правилами, описанными в Техническом стандарте Юникода №10 – Алгоритм сортировки по Юникоду (Unicode® Technical Standard #10 Unicode Collation Algorithm (UTS #10 UCA)). См. *сортировку*.
- NULL values in unique and non-unique indexes. By default, all fields in Tarantool are «NOT NULL». Starting from Tarantool 1.7.6, you can use `is_nullable` option in *space:format()* or *inside an index part definition* to allow storing NULL in indexes. Tarantool partially implements three-valued logic from the SQL standard and allows storing multiple NULL values in unique indexes (gh-1557).
- Sequences and a new implementation of *auto_increment()*. Tarantool 1.7.6 introduces new *sequence number generators* (like CREATE SEQUENCE in SQL). This feature is used to implement new persistent auto increment in spaces (gh-389).
- Vinyl: introduced gap locks in Vinyl transaction manager. The new locking mechanism in Vinyl TX manager reduces the number of conflicts in transactions (gh-2671).
- net.box: *on_connect* and *on_disconnect* triggers (gh-2858).
- Structured logging in *JSON format* (gh-2795).
- (Lua) Lua: *string.strip()* (gh-2785).
- (Lua) added *base64_urlsafe_encode()* to `digest` module (gh-2777).
- Log conflicted keys in master-master replication (gh-2779).
- Allow to disable backtrace in *fiber.info()* (gh-2878).
- Implemented `tarantoolctl rocks make *.spec` (gh-2846).
- Extended the default loader to look for `.rocks` in the parent dir hierarchy (gh-2676).
- `SOL_TCP` options support in *socket:setsockopt()* (gh-598).
- Partial emulation of LuaSocket on top of Tarantool Socket (gh-2727).

Инструменты разработчика:

- Интеграция с IntelliJ IDEA с поддержкой отладки. Появилась возможность использовать IntelliJ IDEA в качестве IDE для разработки и отладки Lua-приложений для Tarantool. См. *Использование IDE*.
- Integration with MobDebug remote Lua debugger (gh-2728).
- Configured `/usr/bin/tarantool` as an alternative Lua interpreter on Debian/Ubuntu (gh-2730).

Новые сторонние библиотеки:

- smtp.client – поддержка SMTP по `libcurl`.

**Версия 1.7.5**

Release type: stable. Release date: 2017-08-22. Tag: 1.7.5. Release: doc-289 or v. 1.7.5.

Данная сборка представляет собой стабильную версию в серии 1.7. Это обновление содержит более 160 исправлений по сравнению с версией 1.7.4.

Изменения или добавления функциональности:

- (Vinyl) a new *force_recovery* mode to recover broken disk files. Use `box.cfg{force_recovery=true}` to recover corrupted data files after hardware issues or power outages (gh-2253).
- (Vinyl) index options can be changed on the fly without rebuild. Now *page_size*, *run_size_ratio*, *run_count_per_level* and *bloom_fpr* index options can be dynamically changed via *index:alter()*. The changes take effect in newly created data files only (gh-2109).
- (Vinyl) improve *box.info.vinyl()* and `index:info()` output (gh-1662).
- (Vinyl) introduce *box.cfg.vinyl_timeout* option to control quota throttling (gh-2014).
- Memtx: stable *index:pairs()* iterators for the TREE index. TREE iterators are automatically restored to a proper position after index's modifications (gh-1796).
- (Memtx) *predictable order* for non-unique TREE indexes. Non-unique TREE indexes preserve the sort order for duplicate entries (gh-2476).
- (Memtx+Vinyl) dynamic configuration of *max tuple size*. Now `box.cfg.memtx_max_tuple_size` and `box.cfg.vinyl_max_tuple_size` configuration options can be changed on the fly without need to restart the server (gh-2667).
- (Memtx+Vinyl) new implementation. Space *truncation* doesn't cause re-creation of all indexes any more (gh-618).
- Extended the *maximal length* of all identifiers from 32 to 65k characters. Space, user and function names are not limited by 32 characters anymore (gh-944).
- *Heartbeat* messages for replication. Replication client now sends the selective acknowledgments for processed records and automatically re-establish stalled connections. This feature also changes *box.info.replication[replica_id].vclock*. to display committed vclock of remote replica (gh-2484).
- Keep track of remote replicas during WAL maintenance. Replication master now automatically preserves xlogs needed for remote replicas (gh-748).
- Enabled *box.tuple.new()* to work without `box.cfg()` (gh-2047).
- *box.atomic(fun, ...)* wrapper to execute function in a transaction (gh-818).
- *box.session.type()* helper to determine session type (gh-2642).
- Hot code *reload* for stored C stored procedures. Use `box.schema.func.reload('modulename.function')` to reload dynamic shared libraries on the fly (gh-910).
- *string.hex()* and `str:hex()` Lua API (gh-2522).
- Package manager based on LuaRocks. Use `tarantoolctl rocks install MODULENAME` to install MODULENAME Lua module from `https://rocks.tarantool.org/`. (gh-2067).
- Lua 5.1 command line options. Tarantool binary now supports „-i“, „-e“, „-m“ and „-l“ command line options (gh-1265).
- Experimental GC64 mode for LuaJIT. GC64 mode allow to operate the full address space on 64-bit hosts. Enable via `-DLUAJIT_ENABLE_GC64=ON compile-time` configuration option (gh-2643).
- Syslog logger now support non-blocking mode. *box.cfg{log_nonblock=true}* now also works for syslog logger (gh-2466).
- Added a VERBOSE *log level* beyond INFO (gh-2467).

- Tarantool now automatically makes snapshots every hour. Please set *box.cfg{checkpoint_interval=0* to restore pre-1.7.5 behavior (gh-2496).
- Increase precision for percentage ratios provided by *box.slab.info()* (gh-2082).
- Stack traces now contain symbols names on all supported platforms. Previous versions of Tarantool didn't display meaningful function names in *fiber.info()* on non-x86 platforms (gh-2103).
- Allowed to create fiber with custom stack size from C API (gh-2438).
- Added `ipc_cond` to public C API (gh-1451).

Новые сторонние библиотеки:

- *http.client* (built-in) - libcurl-based HTTP client with SSL/TLS support (gh-2083).
- *iconv* (built-in) - bindings for iconv (gh-2587).
- authman - API для регистрации пользователя и входа в систему с использованием email и социальных сетей.
- document - хранит вложенные документы в Tarantool.
- synchronized - критические секции для Lua.

**Версия 1.7.4**

Тип версии: предварительная версия. Дата выхода: 2017-05-12. Тег версии: 1.7.4.

Release: v. 1.7.4 or `https://groups.google.com/forum/#!topic/tarantool/3x88ATX9YbY`

Данная сборка представляет собой предварительную версию перед выпуском нового релиза в серии 1.7. Движок vinyl, ключевой компонент 1.7.x, обладает полностью реализованной заявленной функциональностью.

Несовместимые изменения

- Для поддержки vinyl были внесены следующие изменения в параметры `box.cfg()`:
  - переименование `snap_dir` в `memtx_dir`
  - переименование `slab_alloc_arena` (гигабайты) в `memtx_memory` (байты), значение, используемое по умолчанию, изменилось с 1 Гб на 256 МБ
  - переименование `slab_alloc_minimal` в `memtx_min_tuple_size`
  - переименование `slab_alloc_maximal` в `memtx_max_tuple_size`
  - `slab_alloc_factor` больше не используется, не применимо в 1.7.x
  - переименование `snapshot_count` в `checkpoint_count`
  - переименование `snapshot_period` в `checkpoint_interval`
  - переименование `logger` в `log`
  - переименование `logger_nonblock` в `log_nonblock`
  - переименование `logger_level` в `log_level`
  - переименование `replication_source` в `replication`
  - `panic_on_snap_error = true` и `panic_on_wal_error = true` заменены `force_recovery = false`

  Until Tarantool 1.8, you can use deprecated parameters for both initial and runtime configuration, but such usage will print a warning in the server log (gh-1927 and gh-2042).

- Hot standy mode is now off by default. Tarantool automatically detects another running instance in the same `wal_dir` and refuses to start. Use `box.cfg {hot_standby = true}` to enable the hot standby mode (gh-775).
- UPSERT via a secondary key was banned to avoid unclear semantics (gh-2226).
- `box.info` and `box.info.replication` format was changed to display information about upstream and downstream connections ((gh-723):
  - Добавление `box.info.replication[instance_id].downstream.vclock` для отображения последней строки, отправленной на удаленную реплику.
  - Добавление `box.info.replication[instance_id].id`.
  - Добавление `box.info.replication[instance_id].lsn`.
  - Перемещение `box.info.replication[instance_id].{vclock,status,error}` в `box.info.replication[instance_id].upstream.{vclock,status,error}`.
  - Включение всех зарегистрированных реплик из `box.space._cluster` в вывод `box.info.replication`.
  - Переименование `box.info.server.id` в `box.info.id`
  - Переименование `box.info.server.lsn` в `box.info.lsn`
  - Переименование `box.info.server.uuid` в `box.info.uuid`
  - Переименование `box.info.cluster.signature` в `box.info.signature`
  - Возврат значения `nil` вместо `-1` функциями `box.info.id` и `box.info.lsn` во время начальной настройки кластера.
- `net.box`: добавление запрошенные параметров во все запросы:
  - изменение `conn.call(func_name, arg1, arg2,...)` на `conn.call(func_name, {arg1, arg2, ...}, opts)`
  - изменение `conn.eval(func_name, arg1, arg2,...)` на `conn.eval(func_name, {arg1, arg2, ...}, opts)`
- Все запросы поддерживают параметры `timeout = <seconds>``(время задержки в секундах), ``buffer = <ibuf>` (буфер).
- Добавление опции `connect_timeout` в `netbox.connect()`.
- `netbox:timeout()` and `conn:timeout()` are now deprecated. Use `netbox.connect(host, port, { call_16 = true })` for 1.6.x-compatible behavior (gh-2195).
- systemd configuration changed to support `Type=Notify` / `sd_notify()`. Now `systemctl start tarantool@INSTANCE` will wait until Tarantool has started and recovered from xlogs. The recovery status is reported to `systemctl status tarantool@INSTANCE` (gh-1923).
- `log` module now doesn't prefix all messages with the full path to tarantool binary when used without `box.cfg()` (gh-1876).
- `require('log').logger_pid()` was renamed to `require('log').pid()` (gh-2917).
- Removed Lua 5.0 compatible defines and functions (gh-2396):
  - `luaL_Reg` заменяет удаленный `luaL_reg`
  - `lua_objlen(L, i)` заменяет удаленный `luaL_getn(L, i)`
  - Удаление `luaL_setn(L, i, j)` (пустая операция)
  - `luaL_ref(L, lock)` заменяет удаленный `lua_ref(L, lock)`

  - `lua_rawgeti(L, LUA_REGISTRYINDEX, (ref))` заменяет удаленный `lua_getref(L,ref)`
  - `luaL_unref(L, ref)` заменяет удаленный `lua_unref(L, ref)`.
  - `math.fmod()` заменяет удаленный `math.mod()`
  - `string.gmatch()` заменяет удаленный `string.gfind()`

Изменения или добавления функциональности:

- (Vinyl) multi-level compaction. The compaction scheduler now groups runs of the same range into levels to reduce the write amplification during compaction. This design allows Vinyl to support 1:100+ ram:disk use-cases (gh-1821).
- (Vinyl) bloom filters for sorted runs. Bloom filter is a probabilistic data structure which can be used to test whether a requested key is present in a run file without reading the actual file from the disk. Bloom filter may have false-positive matches but false-negative matches are impossible. This feature reduces the number of seeks needed for random lookups and speeds up REPLACE/DELETE with enabled secondary keys (gh-1919).
- (Vinyl) key-level cache for point lookups and range queries. Vinyl storage engine caches selected keys and key ranges instead of entire disk pages like in traditional databases. This approach is more efficient because the cache is not polluted with raw disk data (gh-1692).
- (Vinyl) implemented`). Now all in-memory indexes of a space store pointers to the same tuples instead of cached secondary key index data. This feature significantly reduces the memory footprint in case of secondary keys (gh-1908).
- (Vinyl) new implementation of initial state transfer of JOIN command in replication protocol. New replication protocol fixes problems with consistency and secondary keys. We implemented a special kind of low-cost database-wide read-view to avoid dirty reads in JOIN procedure. This trick wasn't not possible in traditional B-Tree based databases (gh-2001).
- (Vinyl) index-wide mems/runs. Removed ranges from in-memory and and the stop layer of LSM tree on disk (gh-2209).
- (Vinyl) coalesce small ranges. Before dumping or compacting a range, consider coalescing it with its neighbors (gh-1735).
- (Vinyl) implemented transnational journal for metadata. Now information about all Vinyl files is logged in a special `.vylog` file (gh-1967).
- (Vinyl) implemented consistent secondary keys (gh-2410).
- (Memtx+Vinyl) implemented low-level Lua API to create consistent backups. of Memtx + Vinyl data. The new feature provides `box.backup.start()/stop()` functions to create backups of all spaces. *box.backup.start()* pauses the Tarantool garbage collector and returns the list of files to copy. These files then can be copied be any third-party tool, like cp, ln, tar, rsync, etc. `box.backup.stop()` lets the garbage collector continue. Created backups can be restored instantly by copying into a new directory and starting a new Tarantool instance. No special preparation, conversion or unpacking is needed (gh-1916).
- (Vinyl) added statistics for background workers to `box.info.vinyl()` (gh-2005).
- (Memtx+Vinyl) reduced the memory footprint for indexes which keys are sequential and start from the first field. This optimization was necessary for secondary keys in Vinyl, but we optimized Memtx as well (gh-2046).
- LuaJIT was rebased on the latest 2.1.0b3 with out patches (gh-2396):
  - Добавлен бэкенд для JIT-компилятора для архитектуры ARM64
  - Добавлен бэкенд и интерпретатор для JIT-компилятора для архитектуры MIPS64

  - Добавлены некоторые расширения для Lua 5.2 и Lua 5.3
  - Исправление нескольких ошибок
  - Удалены устаревшие функции Lua 5.0 (см. несовместимые изменения выше).

- Запущен новый умный алгоритм хеширования строк в LuaJIT, чтобы избежать замедления работы в случае множества коллизий. Разработали Юрий Соколов (@funny-falcon) и Ник Заварицкий (@mejedi). См. https://github.com/tarantool/luajit/pull/2.
- `box.snapshot()` now updates mtime of a snapshot file if there were no changes to the database since the last snapshot. (gh-2045).
- Implemented `space:bsize()` to return the memory size utilized by all tuples of the space. Contributed by Roman Tokarev (@rtokarev). (gh-2043).
- Новые функции Lua/C вынесены в общедоступный API:
  - `luaT_pushtuple`, `luaT_istuple` (gh-1878)
  - `luaT_error`, `luaT_call`, `luaT_cpcall` (gh-2291)
  - `luaT_state` (gh-2416)
- Exported new Box/C functions to public API: `box_key_def`, `box_tuple_format`, `tuple_compare()`, `tuple_compare_with_key()`. (gh-2225).
- xlogs now can be rotated based on size (`wal_max_size`) as well as the number of written rows (`rows_per_wal`). (gh-173).
- Added `string.split()`, `string.startswith()`, `string.endswith()`, `string.ljust()`, `string.rjust()`, `string.center()` API (gh-2211, gh-2214, gh-2415).
- Added `table.copy()` and `table.deepcopy()` functions. (gh-2212).
- Added `pwd` module to work with UNIX users and groups. (gh-2213).
- Removed noisy «client unix/: connected» messages from logs. Use `box.session.on_connect()`/`on_disconnect()` triggers instead. (gh-1938).

  Триггеры `box.session.on_connect()`/`on_disconnect()`/`on_auth()` также срабатывают для подключений административной консоли.
- tarantoolctl: `eval`, `enter`, `connect` commands now support UNIX pipes (gh-672).
- tarantoolctl: improved error messages and added a new man page (gh-1488).
- tarantoolctl: added filter by `replica_id` to `cat` and `play` commands (gh-2301).
- tarantoolctl: `start`, `stop` and `restart` commands now redirect to `systemctl start/stop/restart` when systemd is enabled (gh-2254).
- net.box: added `buffer = <buffer>` per-request option to store raw MessagePack responses into a C buffer (gh-2195).
- net.box: added `connect_timeout` option (gh-2054).
- net.box: added `on_schema_reload()` hook (gh-2021).
- net.box: exposed `conn.schema_version` and `space.connection` to API (gh-2412).
- log: `debug()`/`info()`/`warn()`/`error()` now doesn't fail on formatting errors (gh-889).
- crypto: added HMAC support. Contributed by Andrey Kulikov (@amdei) (gh-725).

### Версия 1.7.3

Тип версии: бета. Дата выхода: 2016-12-24. Тег версии: 1.7.3-0-gf0c92aa.

Release: v. 1.7.3

Данная сборка представляет собой вторую бета-версию в серии 1.7.

Несовместимые изменения:

- Broken `coredump()` Lua function was removed. Use `gdb -batch -ex "generate-core-file" -p $PID` instead (gh-1886).
- Vinyl disk layout was changed since 1.7.2 to add ZStandard compression and improve the performance of secondary keys. Use the replication mechanism to upgrade from 1.7.2 beta (gh-1656).

Изменения или добавления функциональности:

- Значительный прогресс в стабилизации движка базы данных Vinyl:
  - Исправлены большинство известных отказов системы и ошибок, выдающих плохие результаты.
  - Замена формата всех файлов с данными на XLOG/SNAP.
  - Использование механизма компрессии ZStandard для всех файлов с данными.
  - Сжатие операций UPSERT на лету и объединение горячих клавиш с помощью фонового файбера.
  - Значительное улучшение производительности index:pairs() и index:count().
  - Удаление ненужных конфликтов из транзакций.
  - Уровень In-memory по большей части заменен структурами данных memtx.
  - В большинстве случаев используются специализированные распределители ресурсов.
- Мы все еще активно работаем над Vinyl'ом и планируем добавить многоуровневое слияние и улучшить производительность в работе со вторичными ключами в версии 1.7.4. Это подразумевает изменение формата данных.
- Support for DML requests for space:on_replace() triggers (gh-587).
- UPSERT can be used with the empty list of operations (gh-1854).
- Lua functions to manipulate environment variables (gh-1718).
- Lua library to read Tarantool snapshots and xlogs (gh-1782).
- New `play` and `cat` commands in `tarantoolctl` (gh-1861).
- Улучшена поддержка большого количества активных сетевых клиентов. Проблема #5#1892.
- Support for `space:pairs(key, iterator-type)` syntax (gh-1875).
- Automatic cluster bootstrap now also works without authorization (gh-1589).
- Replication retries to connect to master indefinitely (gh-1511).
- Temporary spaces now work with `box.cfg { read_only = true }` (gh-1378).
- The maximum length of space names increased to 64 bytes (was 32) (gh-2008).

### Версия 1.7.2

Тип версии: бета. Дата выхода: 2016-09-29. Тег версии: `1.7.2-1-g92ed6c4`.

Release: https://groups.google.com/forum/#!topic/tarantool-ru/qUYUesEhRQg or v. 1.7.2

Данная сборка представляет собой версию в серии 1.7.

Несовместимые изменения:

- A new binary protocol command for CALL, which no more restricts a function to returning an array of tuples and allows returning an arbitrary MsgPack/JSON result, including scalars, nil and void (nothing). The old CALL is left intact for backward compatibility. It will be removed in the next major release. All programming language drivers will be gradually changed to use the new CALL (gh-1296).

Изменения или добавления функциональности:

- Разработка движка базы данных Vinyl, наконец, перешла в бета-стадию. В данной версии исправлены более 90 ошибок в Vinyl'е, в частности, удаление непредсказуемых скачков задержки отклика, все известные отказы системы и ошибки, выдающие плохие результаты или их отсутствие.
  - новая архитектура на основе кооперативной многозадачности для устранения скачков задержки отклика,
  - поддержка непоследовательных составных ключей,
  - поддержка вторичных ключей,
  - поддержка `auto_increment()`,
  - типы полей в индексах: number (число), integer (целое число), scalar (скаляр),
  - операции INSERT, REPLACE и UPDATE возвращают новый кортеж, как в memtx'е.
- Мы все еще активно работаем над Vinyl'ом и планируем добавить механизм компрессии `zstd` и новый распределитель ресурсов для Vinyl'а в версии 1.7.3. Это подразумевает изменение формата данных, который планируется внедрить до того, как версия 1.7 станет общедоступной.
- Tab-based autocompletion in the interactive console, `require('console').connect()`, `tarantoolctl enter` and `tarantoolctl connect` commands (gh-86 and gh-1790). Use the TAB key to auto complete the names of Lua variables, functions and meta-methods.
- A new implementation of `net.box` improving performance and solving problems when the Lua garbage collector handles dead connections (gh-799, gh-800, gh-1138 and gh-1750).
- Появилась компрессия снимков memtx и xlog-файлов на лету с использованием быстрого алгоритма компрессии ZStandard. Компрессия настраивается автоматически для получения оптимального соотношения между использованием ЦП и пропускной способностью диска.
- `fiber.cond()` - a new synchronization mechanism for cooperative multitasking (gh-1731).
- Tarantool теперь можно устанавливать из универсальных Snappy-пакетов (`http://snapcraft.io/`) с помощью команды `snap install tarantool --channel=beta`.

Новые модули и пакеты:

- curl - неблокирующие привязки для libcurl
- prometheus - сборщик метрик Prometheus для Tarantool
- gis - полнофункциональное геопространственное расширение для Tarantool
- mqtt - клиент MQTT-протокола для Tarantool
- luaossl - самый полноценный OpenSSL-модуль во вселенной Lua

Устаревшие, удаленные и несовместимые функции:

- `num` and `str` fields type names are deprecated, use `unsigned` and `string` instead (gh-1534).

- `space:inc()` and `space:dec()` were removed (deprecated in 1.6.x) (gh-1289).
- `fiber:cancel()` is now asynchronous and doesn't wait for the fiber to end (gh-1732).
- Implicit error-prone `tostring()` was removed from `digest` API (gh-1591).
- Поддержка SHA-0 (`digest.sha()`) прекращается по причине обновления OpenSSL.
- `net.box` now uses one-based indexes for `space.name.index[x].parts` (gh-1729).
- Бинарный файл Tarantool будет динамически связываться с `libssl.so` во время компиляции вместо загрузки во время выполнения.
- Пакеты Debian и Ubuntu будут использовать встроенную конфигурацию `systemd` вместе с вышедшими из употребления скриптами `sysvinit`.

  В `systemd` появляется возможность управления несколькими экземплярами. Чтобы обновить, выполните следующие действия:

  1. Установите новые пакеты версии 1.7.2.
  2. Убедитесь в наличии файла `ИМЯ_ЭКЗЕМПЛЯРА.lua` в директории `/etc/tarantool/instances.enabled`.
  3. Остановите ЭКЗЕМПЛЯР с помощью `tarantoolctl stop ИМЯ_ЭКЗЕМПЛЯРА`.
  4. Запустите ЭКЗЕМПЛЯР с помощью `systemctl start tarantool@ИМЯ_ЭКЗЕМПЛЯРА`.
  5. Включите ЭКЗЕМПЛЯР во время загрузки системы с помощью `systemctl enable tarantool@ИМЯ_ЭКЗЕМПЛЯРА`.
  6. Введите команду `systemctl disable tarantool; update-rc.d tarantool remove`, чтобы отключить надстройки, совместимые с sysvinit.

  Refer to (gh-1291), comment and *the administration chapter* for additional information.
- Debian and Ubuntu packages start a ready-to-use `example.lua` instance on a clean installation of the package. The default instance grants universe permissions for `guest` user and listens on «localhost:3313».
- Пакеты для Fedora 22 объявлены устаревшими (прекращение поддержки).

**Версия 1.7.1**

Тип версии: альфа. Дата выхода: 2016-07-11.

Release: https://groups.google.com/forum/#!topic/tarantool/KGYj3VKJKb8 or v. 1.7.1

Данная сборка представляет собой первую альфа-версию в серии 1.7. Основной функцией данной версии является новый движок базы данных под названием «vinyl». Vinyl представляет собой оптимизированный для записи движок базы данных, который позволяет сохранять объем сохраняемых данных, превышающий объем доступной памяти в 10-100 раз. Vinyl является продолжением движка Sophia из версии 1.6, а именно ответвлением и дальним родственником Sophia Дмитрия Симоненко. Новый Vinyl заменяет Sophia. Он реализован в виде журнально-структурированного дерева со слиянием (log-structured merge tree – LSM-tree). Однако усовершенствование таких традиционных недостатков журнально-структурированных хранилищ, как низкая производительность при чтении и непредсказуемая задержка во времени при записи, стоит больших усилий. Отдельный индекс секционирован по диапазонам между многими структурами данных LSM, в каждой из который находятся собственные буферы оперативной памяти регулируемого размера. Секционирование по диапазонам позволяет осуществить слияние LSM-уровней, чтобы добиться большей детализации, а также отдать приоритет горячим диапазонам по отношению к холодным в том, что касается доступа к ресурсам, таким как оперативная память и ввод-вывод. Планировщик слияний предназначен для сведения времени задержки записи к минимуму, а также для поддержания производительности при чтении в приемлемых пределах. На сегодняшний день Vinyl поддерживает только первичные индексы. Индекс может состоять из

256 частей, как в MemTX'е, по сравнению с 8 в Sophia. Поддерживает чтение по компонентам ключа. Вскоре ожидается поддержка непоследовательных составных ключей, а также вторичных ключей. Наше намерение заключается в том, чтобы убрать любые ограничения, которые есть сейчас в Vinyl'е, чтобы сделать его полноценным компонентом Tarantool.

Изменения или добавления функциональности:

- Дисковый движок, который в более ранних версиях Tarantool назывался `sophia` или `phia`, заменен новым движком под названием `vinyl`.
- Добавлены новые типы индексируемых полей.
- Обновлена версия LuaJIT.
- Поддерживается автоматическая настройка набора реплик, что существенно упрощает настройку нового набора реплик.
- Функция `space_object:inc()` объявлена устаревшей.
- Функция `space_object:dec()` объявлена устаревшей.
- Добавлена функция `space_object:bsize()`.
- Удалена функция `box.coredump()`, аналог см. в главе *Создание дампов памяти*.
- Добавлена опция настройки `hot_standby` (горячий резерв).
- Исправленные или переименованные конфигурационные параметры:
  - `slab_alloc_arena` (в гигабайтах) в `memtx_memory` (в байтах),
  - `slab_alloc_minimal` в `memtx_min_tuple_size`,
  - `slab_alloc_maximal` в `memtx_max_tuple_size`,
  - `replication_source` в `replication`,
  - `snap_dir` в `memtx_dir`,
  - `logger` в `log`,
  - `logger_nonblock` в `log_nonblock`,
  - `snapshot_count` в `checkpoint_count`,
  - `snapshot_period` в `checkpoint_interval`,
  - `panic_on_wal_error` и `panic_on_snap_error` объединены в `force_recovery`.
- В версиях Tarantool до 1.8 можно использовать *устаревшие параметры* как для начальной, так и для рабочей конфигурации, но в таком случае Tarantool выдаст предупреждение. Также можно указывать как устаревшие, так и новые параметры при условии, что их значения согласованы. В противном случае, Tarantool выдаст ошибку.
- У кластера репликации появилась возможность автоматической настройки, что существенно упрощает настройку нового кластера.
- Новые индексируемые типы данных: INTEGER (целое число) и SCALAR (скаляр).
- Рефакторинг кода и улучшение производительности.
- LuaJIT обновлен до версии 2.1-beta116.

### Tarantool 1.6

Release: v. 1.6.0

**Версия 1.6.9**

Release type: maintenance. Release date: 2016-09-27. Release tag: 1.6.9-4-gcc9ddd7. Release: v. 1.6.9

Since February 15, 2017, due to Tarantool issue (gh-2040) there no longer is a storage engine named `sophia`. It will be superseded in version 1.7 by the `vinyl` storage engine.

Несовместимые изменения:

- Поддержка SHA-0 (`digest.sha()`) прекращается по причине обновления OpenSSL.
- Бинарный файл Tarantool будет динамически связываться с libssl.so во время компиляции вместо загрузки во время выполнения.
- Пакеты для Fedora 22 объявлены устаревшими (прекращение поддержки).

Изменения или добавления функциональности:

- Tab-based autocompletion in the interactive console (gh-86).
- LUA_PATH and LUA_CPATH environment variables taken into account, like in PUC-RIO Lua (gh-1428).
- Search for `.dylib` as well as for `.so` libraries in OS X (gh-810).
- A new `box.cfg { read_only = true }` option to emulate master-slave behavior (gh-246).
- `if_not_exists = true` option added to `box.schema.user.grant` (gh-1683).
- `clock_realtime()`/`monotonic()` functions added to the public C API (gh-1455).
- `space:count(key, opts)` introduced as an alias for `space.index.primary:count(key, opts)` (gh-1391).
- Upgrade script for 1.6.4 -> 1.6.8 -> 1.6.9 (gh-1281).
- Support for OpenSSL 1.1 (gh-1722).

Новые модули и пакеты:

- curl - неблокирующие привязки для libcurl
- prometheus - сборщик метрик Prometheus для Tarantool
- gis – полнофункциональное геопространственное расширение для Tarantool.
- mqtt – клиент MQTT-протокола для Tarantool
- luaossl - самый полноценный OpenSSL-модуль во вселенной Lua

**Версия 1.6.8**

Release type: maintenance. Release date: 2016-02-25. Release tag: 1.6.8-525-ga571ac0. Release: v. 1.6.8

Несовместимые изменения:

- RPM-пакеты для CentOS 7 / RHEL 7 Fedora 22+ будут использовать встроенную конфигурацию systemd без устаревших скриптов sysvinit. В systemd появляется возможность управления несколькими экземплярами. Чтобы обновить, выполните следующие действия:

  1. Убедитесь в наличии файла `ИМЯ_ЭКЗЕМПЛЯРА.lua` в директории `/etc/tarantool/instances.available`.
  2. Остановите ЭКЗЕМПЛЯР с помощью `tarantoolctl stop ИМЯ_ЭКЗЕМПЛЯРА`.

3. Запустите ЭКЗЕМПЛЯР с помощью `systemctl start tarantool@ИМЯ_ЭКЗЕМПЛЯРА`.
4. Enable INSTANCENAME during system boot using `systemctl enable tarantool@INTANCENAME`.

Директория `/etc/tarantool/instance.enabled` больше не используется для платформ, запускаемых по `systemd`.

Для получения дополнительной информации см. главу *по администрированию серверной части*.

- Sophia was upgraded to v2.1 to fix upsert, memory corruption and other bugs. Sophia v2.1 doesn't support old v1.1 data format. Please use Tarantool replication to upgrade (gh-1222).
- Ubuntu Vivid, Fedora 20, Fedora 21 объявлены устаревшими по причине прекращения поддержки.
- i686-пакеты объявлены устаревшими. Используйте наши спецификации по RPM и DEB для сборки на своей инфраструктуре.
- Обновите `yum.repos.d` и/или apt `sources.list.d` в соответствии с инструкциями по ссылке `http://tarantool.org/download.html`

Изменения или добавления функциональности:

- Tarantool 1.6.8 fully supports ARMv7 and ARMv8 (aarch64) processors. Now it is possible to use Tarantool on a wide range of consumer devices, starting from popular Raspberry PI 2 to coin-size embedded boards and no-name mini-micro-nano-PCs (gh-1153). (Also qemu works well, but we don't have real hardware to check.)
- Tuple comparator functions were optimized, providing up to 30% performance boost when an index key consists of 2, 3 or more parts (gh-969).
- Tuple allocator changes give another 15% performance improvement (gh-1298).
- Replication relay performance was improved by reducing the amount of data directory re-scans (gh-1150).
- A random delay was introduced into snapshot daemon, reducing the chance that multiple instances take a snapshot at the same time (gh-732).
- Движок базы данных Sophia был обновлен до версии 2.1:
  - изоляция сериализуемых снимков (SSI – Serializable Snapshot Isolation),
  - режим хранения в оперативной памяти,
  - режим хранения без кэша,
  - режим хранения в кэше с подключением к базе данных,
  - внедренный AMQ-фильтр,
  - режим LRU (удаление страниц, которые дольше всего не использовались),
  - отдельная компрессия горячих и холодных данных,
  - внедрение снимков для быстрого восстановления,
  - реорганизация и исправление ошибок в upsert,
  - новые метрики производительности.

  Обратите внимание на «Несовместимые изменения» выше.
- Allow to remove servers with non-zero LSN from `_cluster` space (gh-1219).
- `net.box` now automatically reloads space and index definitions (gh-1183).
- The maximal number of indexes in space was increased to 128 (gh-1311).

- New native `systemd` configuration with support of instance management and daemon supervision (CentOS 7 and Fedora 22+ only). Please note «Incompatible changes» above (gh-1264).
- Пакет Tarantool принят в официальный репозиторий Fedora (https://apps.fedoraproject.org/packages/tarantool).
- Пакет Tarantool (OS X) принят в официальный репозиторий Homebrew (https://formulae.brew.sh/formula/tarantool).
- Clang compiler support was added on FreeBSD. (gh-786).
- Support for musl libc, used by Alpine Linux and Docker images, was added (gh-1249).
- Добавлена поддержка GCC 6.0.
- Получили поддержку Ubuntu Wily, Xenial и Fedora 22, 23 и 24, для которых мы создаем официальные пакеты.
- box.info.cluster.uuid can be used to retrieve cluster UUID (gh-1117).
- Многочисленные исправления в документации, добавлена документация по пакетам `syslog`, `clock`, `fiber.storage`, встроенное практическое задание получило обновление.

Новые модули и пакеты:

- Tarantool перешел на новую облачную инфраструктуру на основе Docker. Новый инструмент интеграции разработки buildbot значительно уменьшает время передачи коммитов в пакеты. Официальные репозитории по ссылке http://tarantool.org теперь содержат последнюю версию сервера, модулей и коннекторов. См. http://github.com/tarantool/build
- Репозитории по ссылке http://tarantool.org/download.html were был перенесены в облачное хранилище http://packagecloud.io (при поддержке Amazon AWS). Благодарим packagecloud.io за поддержку свободного ПО!
- `memcached` – внедрение текстового и бинарного протокола memcached для Tarantool. Превращает Tarantool в memcached с доступом к базе данных с репликацией по схеме мастер-мастер. См. https://github.com/tarantool/memcached
- `migrate` – модуль Tarantool для миграции с версии 1.5 на версию 1.6. См. https://github.com/bigbes/migrate
- `cqueues` – асинхронный Lua-каркас для работы по сети с потоками и уведомлениями (разработал @daurnimator). Проблема 1204.

**Версия 1.6.7**

Release type: maintenance. Release date: 2015-11-17. Release: v. 1.6.7 Incompatible changes:

- Изменился синтаксис команды `upsert`, и из нее был удален дополнительный аргумент `key`. Первичный ключ для поиска всегда берется из кортежа, который является вторым аргументом в upsert. `upsert()` добавили довольно поздно в рабочем цикле, и в проекте была очевидная ошибка, которую нам пришлось исправлять. Извините.
- Функцию `fiber.channel.broadcast()` удалили, потому что ее никто не использовал, и она работала некорректно.
- Команда `reload` утилиты tarantoolctl переименована в``eval``.

Изменения или добавления функциональности:

- Опция `logger` допускает синтаксис для вывода в системный журнал syslog. Используйте синтаксис URI, чтобы определить место назначения журнала: в файл, в конвейер или syslog.
- `replication_source` принимает массив URI, так что в каждой реплике может быть до 30 узлов.

- RTREE-индекс принимает два типа функций `distance`: `euclid` и `manhattan`.
- `fio.abspath()` – новая функция в модуле `fio` для конвертации относительного пути в абсолютный.
- Название процесса теперь можно определить с помощью встроенного модуля `title`.
- В данной версии используется LuaJIT 2.1.

Новые сторонние библиотеки:

- `memcached` помогает Tarantool понимать бинарный протокол Memcached. Поддержка текстового протокола находится в процессе разработки и будет добавлена в отдельный модуль без изменений основных компонентов.

**Версия 1.6.6**

Release type: maintenance. Release date: 2015-08-28. Release: v. 1.6.6

Tarantool версии 1.6 больше не получает значимых новых функций, но продолжает поддерживаться. Разработчики сосредоточили свои усилия на версии 1.9.

Несовместимые изменения:

- Появляется новая схема системного спейса `_index` для размещения многомерных RTREE-индексов. Tarantool 1.6.6 нормально работает со старыми снимками и системными спейсами, но нельзя будет запустить Tarantool версии 1.6.5 с директорий, созданной в Tarantool версии 1.6.6, как нельзя будет ввести запрос в Tarantool 1.6.6 с net.box версии 1.6.5.
- Переименование `box.info.snapshot_pid` в `box.info.snapshot_in_progress`

Изменения или добавления функциональности:

- Потоковая архитектура для работы по сети. Сетевой ввод-вывод окончательно переведен на отдельный поток, что увеличит производительность отдельного экземпляра до 50%.
- Потоковая архитектура для создания контрольных точек. Tarantool больше не делает ответвлений для создания снимка, а использует отдельный поток, получая доступ к данным с помощью вида постоянного просмотра. Это помогает устранить скачки задержки отклика во время создания снимков.
- Хранимые процедуры на языках C/C++. Хранимые процедуры на языках C/C++ дают скорость (в 3-4 раза больше по сравнению с Lua-версией по нашим подсчетам), а также возможность неограниченного расширения. Поскольку процедуры C/C++ выполняются там же, где располагается база данных, они могут с легкостью повредить базу данных. См. *API для языка C*.
- Многомерный RTREE-индекс. RTREE-индекс теперь поддерживает большое количество измерений (до 32). Структура данных RTREE была оптимизирована так, чтобы действительно использовать R*-TREE. Мы работаем над дальнейшим улучшением индекса, в частности, над функцией конфигурации расстояния. См. https://github.com/tarantool/tarantool/wiki/R-tree-index-quick-start-and-usage
- Sophia 2.1.1 с поддержкой компрессии и составных первичных ключей. См. https://groups.google.com/forum/#!topic/sophia-database/GfcbEC7ksRg
- New `upsert` command available in the binary protocol and in stored functions. The key advantage of upsert is that it's much faster with write-optimized storage (sophia storage engine), but some caveats exists as well. See (gh-905) for details. Even though upsert performance advantage is most prominent with sophia engine, it works with all storage engines.
- Better memory diagnostics information for fibers, tuple and index arena. Try a new command `box.slab.stats()`, for detailed information about tuple/index slabs, `fiber.info()` now displays information about memory used by the fiber.

- Операции update и delete работают с использованием вторичного индекса, если индекс уникальный.
- Триггеры для аутентификации. Установите триггеры `box.session.on_auth` для отслеживания событий аутентификации. API для триггеров улучшили, чтобы он отображал все заданные триггеры, старые триггеры легко удалить.
- Разнообразные улучшения производительности встроенного модуля `net.box`.
- Оптимизация производительности BITSET-индекса.
- `panic_on_wal_error` представляет собой динамический параметр конфигурации.
- Поле iproto `sync` доступно в Lua как `session.sync()`.
- `box.once()` – новый метод для вызова кода однократно в течение срока жизни экземпляра и набора реплик. Используйте `once()` для настройки спейсов и пользователей, а также для обновления схемы в эксплуатационной среде.
- `box.error.last()` возвращает последнюю ошибку в сессии.

Новые сторонние библиотеки:

- Следующие модули LuaJIT 2.0 теперь являются встроенными: `jit.*`, `jit.dump`, `jit.util`, `jit.vmdef`. См. http://luajit.org/ext_jit.html
- `strict` – встроенный пакет, который запрещает использование необъявленных переменных в Lua. Работа ведется в таком режиме, когда Tarantool компилируется с отладкой. Чтобы включить/отключить этот режим, используйте `require('strict').on()`/`require('strict').off()` соответственно.
- `pg` и `mysql` – модули, доступные по ссылке http://rocks.tarantool.org – работают с MySQL и PostgreSQL из Tarantool.
- `gperftools` rock, available at http://rocks.tarantool.org - getting performance data using Google's gperf from Tarantool.
- `csv` – встроенный модуль для разбора и загрузки данных в формате CSV (значения, разделенные запятыми).

Поддержка новой платформы:

- Fedora 22, Ubuntu Vivid

### 11.1.5 Enterprise SDK changelog

#### Versioning policy

A *Tarantool Enterprise SDK* version consists of two parts:

```
<TARANTOOL_BASE_VERSION>-r<REVISION>
```

For example: `2.11.1-0-gc42d9735b-r589`.

- `TARANTOOL_BASE_VERSION` is the Community version which the Enterprise version is based on.
- `REVISION` is the SDK revision. Besides Tarantool itself, it includes the `tt` utility, a set of open and closed source modules, and examples. Learn more from *Package contents*.

### 616

- Updated crud to 1.4.3.
- Updated luatest to 1.0.1.
- Updated migrations to 0.7.0.
- Updated tt-ee to 2.1.2.

### 613

- Updated cartridge to 2.8.5.
- Updated crud to 1.4.2.
- Added frontend-core 8.2.2.
- Updated membership to 2.4.2.
- Updated sideservice to 0.2.0.
- Updated tt-ee to 2.1.1.
- Updated vshard to 0.1.26.

### 609

- Updated httpgo to 0.2.1.
- Added httpgo-crud 0.1.0.
- Updated `tarantool-2.11` to 2.11.2.

### 606

- Updated `tarantool-master` to 3.0.0-beta1.

### 605

- Updated cartridge to 2.8.4.
- Updated crud to 1.4.1.
- Updated ddl to 1.6.5.
- Added httpgo 0.2.0.
- Updated tt-ee to 2.0.0.

#### 598

- Updated cartridge-cli to 2.12.9.
- Updated tt-ee to 1.3.1.

#### 595

- Updated tt-ee to 1.3.0.
- Updated cartridge to 2.8.3.
- Updated cartridge-cli-extensions to 1.1.2.
- Updated crud to 1.3.0.
- Updated queue to 1.3.3.
- Updated sharded-queue to 0.1.1.
- Updated membership to 2.4.1.
- Added tests for Astra Linux 1.7.

#### 589

- Updated `tarantool-2.10` to 2.10.8.
- Updated `tarantool-master` to 3.0.0-alpha3.
- Updated migrations to 0.6.0.
- Updated tt-ee to 1.2.0.
- Updated space-explorer to 1.1.8.
- Updated cartridge-metrics-role to 0.1.1.
- Updated cartridge to 2.8.2.
- Updated expirationd to 1.5.0.
- Added sideservice 0.1.0.

#### 579

- Updated cartridge-cli to 2.12.7.
- Updated `tarantool-2.11` to 2.11.1.

### 577

- Added crud 1.2.0.
- Added ddl 1.6.3.
- Added sharded-queue 0.1.0.
- Added ddl 1.6.4.
- Updated tt-ee to 1.1.2.
- Updated cartridge-cli to 2.12.6.

### 563

- Updated `tarantool-2.10` to 2.10.7.
- Updated `tarantool-2.11` to 2.11.0.
- Added kafka 1.6.6.
- Added vshard 0.1.24.
- Added metrics 1.0.0.
- Added cartridge-metrics-role 0.1.0.
- Added cartridge 2.8.0.
- Added http 1.5.0.

### 557

- Added checks 3.3.0.
- Updated cartridge-cli to 2.12.5.

### 553

- Added `tt-ee` and `tt` environment configuration.
- Added crud 1.1.1.
- Added avro-schema 3.1.1.
- Added expirationd 1.4.0.
- Added graphql 0.3.0.
- Added graphqlapi 0.0.10.
- Added metrics 0.17.0.
- Added migrations 0.5.0.
- Added oracle 1.4.0.
- Added cartridge 2.7.9.
- Added vshard 0.1.23.
- Added kafka 1.6.5.

#### 549

- Updated `tarantool-2.10` to 2.10.6.

#### 545

- Updated `tarantool-2.11` to 2.11.0-rc2.

#### 543

- Added the `tarantool-2.11` submodule.

#### 542

- Updated `tarantool-1.10` to 1.10.15.

#### 541

- Updated `tarantool-master` to `3.0.0-entrypoint`.

#### 540

- Updated `tarantool-2.10` to 2.10.5.

#### 539

- Added vshard 0.1.22.

#### 538

- Updated `tarantool-2.8` to apply 2 hotfixes.

#### 537

- Fix non-interactive installation of the `brew` package.
- Changed the owner of the `/usr/local/bin` directory.
- Installed `awscli@1` instead of `awscli` since it takes much less time.

### 536

- Added the missing property `2.10` for scope `CACHE` in CMakeLists.txt.

### 535

- Added expirationd 1.3.1.

### 534

- Added crud 1.0.0.

### 533

- Use runners with label `regular` for builds and the tagged release workflow.

### 532

- Added http 1.4.0.
- Added space-explorer 1.1.7.
- Added checks 3.2.0.
- Added metrics 0.16.0.
- Added cartridge 2.7.8.

### 531

- Added the `-DENABLE_LTO=ON` flag for tarantool-ee@master branch to CMakeLists.txt

### 530

- Upgraded devtoolset from 8 to 9. It was required for upgrading ld from 2.30 to 2.31+ for LTO.

### 529

- Updated tarantool's master branch to a recent revision.

### 528

- Fixed code style in the Linux and macOS workflows.

#### 527

- Reliably install packages in macOS builds.

#### 526

- Refactored the way that GC64 builds are defined in the build workflow. There are no changes to the composition of resulting bundles.

#### 525

- Added alerting failures in builds on stable branches and integration testing to VK Teams chats.

#### 524

- Updated to fresh tarantool master (`2.11.0-entrypoint-107-ga18449d`)

#### 523

- Added cartridge 2.7.7.

#### 522

- Outdated workflow runs are now canceled to save CI time.

#### 521

- Added crud 0.14.1.
- Added expirationd 1.3.0.
- Added metrics 0.15.1.
- Added queue 1.2.2.

#### 520

Release SDK by tags:

- Run workflow in SDK docker container.
- Upload SDK files for 1.10, 2.8, 2.10 versions to release folder.
- Add consistency check for all versions.

#### 519

- On feature branches, SDK is now rebuilt only on relevant changes.

#### r518

- Added frontend core 8.2.1.
- Added vshard 0.1.21.
- Added http 1.3.0.
- Added cartridge 2.7.6.

#### r517

- Updated Tarantool EE to 2.10.4.

#### r516

- Updated bundled OpenSSL to version 1.1.1q.

#### r515

- Removed support of Tarantool 2.7.
- Started using `tarantool/actions/prepare-checkout` to make builds more stable.

#### r514

- Remove the local registry and setup using GitHub registry.
- Sync rocks cache to s3 and back.
- Setup using shared runners.
- Refactor and format `ci-linux.yml` and `ci-macos.yml`.

#### r513

- Removed kafka 1.5.0 due to a build issue with Tarantool 2.10.3 and higher.
- Updated kafka to version 1.6.2.

#### r512

- Updated tuple-keydef to version 0.0.3.

#### r511

- Enabled parallel build of rocks for macOS in CI.

#### r510

- Updated Tarantool to *2.10.3*.
- Added a readable error for the case when the flight recoder fails to write data due to insufficient free space on the disk device. Previously, it was sending a `SIGBUS` error (gh-196).
- Fixed a crash in the flight recorder caused by non-thread-safe log recording from multiple threads (gh-226).

#### r502

- Updated Tarantool to *2.10.2*.
- Increased resolution of stored entries in flight recorder (gh-193).
- Fixed a bug in the flight recorder that resulted in skipping log entries in case `box.cfg.log_level` is less than `flightrec_log_level` (gh-201).

#### r498

- Updated Tarantool to *2.10.1*.
- Updated Cyrus SASL to version 2.1.28.
- Updated OpenLDAP to version 2.5.13.
- Updated LZ4 to version 1.9.3. Fixed CVE-2021-3520.
- Fixed replication reconnect failure after disabling SSL encryption (gh-137).
- Fixed a crash that occurred while tyring to start an instance that has a compressed memtx space (gh-171).
- Fixed CVE-2022-29242 in GOST SSL engine.
- Fixed a bug in the flight recorder reader implementation that resulted in a hang or error while trying to open an empty section (gh-187).

#### r467

##### Breaking changes

- Default audit log format was changed to CSV.

##### Functionality added or changed

###### Enterprise

- Implemented user-defined audit events. Now it's possible to log custom messages to the audit log from Lua (gh-65).
- **[Breaking change]** Switched the default audit log format to CSV. The format can be switched back to JSON using the new `box.cfg.audit_format` configuration option (gh-66).
- Implemented the audit log filter. Now, it's possible to enable logging only for a subset of all audit events using the new `box.cfg.audit_filter` configuration option (gh-67).

###### Core

- Implement constraints and foreign keys. Now a user can create function constraints and foreign key relations (gh-6436).
- Changed log level of some information messages from critical to info (gh-4675).
- Added predefined system events: `box.status`, `box.id`, `box.election` and `box.schema` (gh-6260).
- Introduced transaction isolation levels in Lua and IPROTO (gh-6930).

###### Vinyl

- Disabled the deferred DELETE optimization in Vinyl to avoid possible performance degradation of secondary index reads. Now, to enable the optimization, one has to set the `defer_deletes` flag in space options (gh-4501).

###### Lua

- Added support of console autocompletion for net.box objects `stream` and `future` (gh-6305).

###### Datetime

- **Parse method to allow converting string literals in extended iso-8601** or rfc3339 formats (gh-6731).
- **The range of supported years has been extended in all parsers to cover** fully -5879610-06-22..5879611-07-11 (gh-6731).

### Build

- Added bundling of *GNU libunwind* to support backtrace feature on *AARCH64* architecture and distributives that don't provide *libunwind* package.
- Re-enabled backtrace feature for all *RHEL* distributions by default, except for *AARCH64* architecture and ancient *GCC* versions, which lack compiler features required for backtrace (gh-4611).

### Bugs fixed

#### Enterprise

- Disabled audit log unless explicitly configured (gh-39). Before this change, audit events were written to stderr if `box.cfg.audit_log` wasn't set. Now, audit log is disabled in this case.
- Disabled audit logging of replicated events (gh-59). Now, replicated events (for example, user creation) are logged only on the origin, never on a replica.

#### Core

- Banned DDL operations in space on_replace triggers, since they could lead to a crash (gh-6920).
- Fixed a bug due to which all fibers created with `fiber_attr_setstacksize()` leaked until the thread exit. Their stacks also leaked except when `fiber_set_joinable(..., true)` was used.
- Fixed a crash in mvcc connected with secondary index conflict (gh-6452).
- Fixed a bug which resulted in wrong space count (gh-6421).
- Select in RO transaction now reads confirmed data, like a standalone (autocommit) select does (gh-6452).

#### Replication

- Fixed potential obsolete data write in synchronious replication due to race in accessing terms while disk write operation is in progress and not yet completed.
- Fixed replicas failing to bootstrap when master is just re-started (gh-6966).

#### Lua

- Fixed the behavior of tarantool console on SIGINT. Now Ctrl+C discards the current input and prints the new prompt (gh-2717).

#### Triggers

- Fixed assertion or segfault when MP_EXT received via net.box (gh-6766).
- Now ROUND() properly support INTEGER and DECIMAL as the first argument (gh-6988).

#### Datetime

- Intervals received after datetime arithmetic operations may be improperly normalized if result was negative

```
tarantool> date.now() - date.now()
---
- -1.000026000 seconds
...
```

  I.e. 2 immediately called `date.now()` produce very close values, whose difference should be close to 0, not 1 second (gh-6882).

#### Net.box

- Changed the type of the error returned by net.box on timeout from ClientError to TimedOut (gh-6144).

#### r457

- Fixed some binary protocol encryption bugs.

#### r455

- Added *binary protocol encryption*.
- Added *tuple field compression*.

### 11.1.6 Tarantool release policy

#### Введение

Релизная политика Tarantool меняется, чтобы стать более понятной. Вводится новый формат версионирования, похожий на семантическое версионирование (SemVer), а также новый жизненный цикл версий, предполагающий больше серий с долгосрочной поддержкой. В этом документе подробно рассказано о новой релизной политике, правилах версионирования и жизненном цикле *релизных серий*.

Новая релизная политика заменяет *старую* для следующих серий:

- `2.x.y`, начиная с `2.10.0`. Первая версия в этой серии — `2.10.0-beta1`.
- В будущем — `3.0.0`.

Вот самые важные нововведения:

- Третья часть номера версии больше не указывает на тип версии (альфа-, бета- или релизная). Если это число отлично от нуля, то версия является патчем и содержит только исправление ошибок. На конце номеров предварительных версий теперь есть соответствующие суффиксы, например: `3.0.0-alpha1`.
- Согласно старой релизной политике `1.10` была серией с долгосрочной поддержкой (long-term support, LTS), а серия `2.x.y` — нет, хоть и содержала стабильные релизы. Теперь долгосрочной поддержкой пользуются обе серии.

Ниже вы найдете подробные сведения о новой политике версионирования.

### Политика версионирования

#### Релизные серии и версии

В основе новой релизной политики Tarantool лежит поддержка нескольких релизных серий. У каждой серии свой жизненный цикл, и в нее входят как предварительные, так и релизные версии.

**Что такое релизная серия?** Релизная серия — это ряд версий для разработки и производственной эксплуатации. Эта последовательность версий линейно развивается по определенному плану. У каждой серии свой жизненный цикл, а версии в ней обладают некоторой гарантией совместимости друг с другом и с версиями других серий. Срок поддержки каждой серии — минимум два года с выхода первого релиза.

**Что такое релизная версия?** Релизная версия, или релиз — это тщательно протестированный и готовый к производственной эксплуатации дистрибутив Tarantool, привязанный к определенному коммиту. Номер релиза состоит из трех частей:

```
MAJOR.MINOR.PATCH
```

Каждое число соответствует одному из трех типов релизных версий, о чем подробно рассказано ниже.

**Мажорный релиз** Мажорная релизная версия — это первый *релиз* в своей *релизной серии*. Релиз включает в себя новую функциональность и может нарушать обратную совместимость с прежними версиями. Каждый следующий мажорный релиз получает номер, в котором изменено первое число:

```
MAJOR.0.0

3.0.0
```

**Минорный релиз** В минорном релизе могут вводиться новые функциональные возможности. Тем не менее он гарантированно будет совместим с более ранними версиями серии. Могут быть также исправлены ошибки. Каждый следующий минорный релиз получает номер, в котором изменено второе число:

```
MAJOR.MINOR.0

3.1.0
3.2.0
```

**Патч** Патч (patch release) исправляет ошибки предыдущих версий, но не добавляет новую функциональность. Каждый следующий патч получает номер, в котором изменено третье число:

```
MAJOR.MINOR.PATCH

3.0.1
3.0.2
```

Релизы отвечают ряду требований:

- Каждый релиз предварительно тестировался и обкатывался во внутренних проектах, пока не исчезли все сомнения в его стабильности.
- Исправлены все обнаруженные ошибки в стандартных сценариях использования.
- Нет регрессии, то есть нет новых ошибок в функциональности из прошлых релизов. В случае мажорного релиза, нет регрессии в функциональности из предыдущей серии.

Backwards compatibility is guaranteed between all versions in the same release series. It is also appreciated, but not guaranteed between different release series (major number changes). See *compatibility guarantees page* for details.

### Предварительные версии

**Что такое предварительная версия?** Предварительными называются версии, опубликованные для тестирования и обкатки, но не предназначенные для производственной эксплуатации. Эти версии нумеруются по тому же принципу, что и релизы, но имеют суффикс:

```
MAJOR.MINOR.PATCH-suffix
```

Ниже подробно рассказывается о четырех типах предварительных версий.

**Dev-сборка** Dev-сборка (development build) отражает определенный этап разработки продукта. Такие сборки используются только в процессе разработки и внутреннего тестирования.

Dev-сборки обозначаются суффиксом, полученным командой `$(git describe --always --long)-dev`:

```
MAJOR.MINOR.PATCH-describe-dev

2.10.2-149-g1575f3c07-dev
3.0.0-alpha1-14-gxxxxxxxxx-dev
3.0.0-entrypoint-17-gxxxxxxxxx-dev
3.1.2-5-gxxxxxxxxx-dev
```

**Альфа-версия** В альфа-версию (alpha version) включена часть функциональных возможностей, запланированных в релизной серии. Альфа-версия может быть неполной или нестабильной, а также не обладать обратной совместимостью с предыдущей серией.

Альфа-версии предназначены для пользователей, готовых попробовать новую серию на раннем этапе, а также для тех, кто разрабатывает зависимые компоненты — коннекторы и модули.

```
MAJOR.MINOR.PATCH-alphaN

3.0.0-alpha1
3.0.0-alpha2
```

**Бета-версия** В бета-версию (beta version) включены все функциональные возможности, запланированные в релизной серии. Бета-версия подходит, чтобы начать разработку нового приложения.

Бета-версия позволяет проверить, насколько готова к выпуску та или иная функциональная возможность. На основании этого принимается решение, доработать ли эту возможность, удалить или заменить. Бета-версия может содержать известные ошибки новой функциональности, а также регрессионные ошибки в распространенных сценариях использования.

```
MAJOR.MINOR.PATCH-betaN

3.0.0-beta1
3.0.0-beta2
```

Note that the development of `2.10.0`, the first release under the new policy, starts with version `2.10.0-beta1`.

**Релиз-кандидат** В релиз-кандидате (release candidate, RC) исправляются ошибки и совершенствуется функциональность. Он также помогает собирать отзывы перед публикацией релиза. Набор функциональных возможностей в релиз-кандидате такой же, как в бета-версии. Известные ошибки в типичных сценариях использования устранены, регрессионных ошибок нет.

Релиз-кандидат подходит для использования в предпроизводственной среде (staging).

```
MAJOR.MINOR.PATCH-rcN

3.0.0-rc1
3.0.0-rc2
3.0.1-rc1
```

### Жизненный цикл релизной серии

Каждая релизная серия проходит три стадии:

- *Ранняя разработка*
- *Поддержка*
- *Прекращение поддержки*

### Ранняя разработка

Стадия ранней разработки (early development) продолжается до выхода *мажорного релиза*. На этой стадии публикуются альфа- и бета-версии, а также релиз-кандидаты.

Стадия разделена на два этапа:

1. Разработка и проверка новой функциональности в альфа- и бета-версиях. На этом этапе функциональные возможности могут добавляться и в некоторых случаях удаляться.
2. Стабилизация. С момента выхода первого релиз-кандидата набор функциональных возможностей не меняется.

### Поддержка

Эта стадия начинается, когда выходит первый релиз. Теперь в релизной серии могут появляться только изменения, не нарушающие обратную совместимость.

На этой стадии устраняются известные проблемы безопасности и исправляются регрессионные ошибки, возникшие в серии.

Патчи для регрессионных и других ошибок выпускаются, пока серия не перейдет со стадии поддержки на стадию прекращения поддержки (end of life, EOL).

Решение о том, исправить ли конкретную ошибку в релизной серии, принимается в зависимости от того, насколько проблема серьезна, сложно ли будет ее устранить в предыдущих версиях и нарушит ли исправление обратную совместимость.

На этой стадии в релизной серии могут выпускаться новые функциональные возможности, но все они будут обратно совместимы с предыдущими версиями. Кроме того, может быть выпущен релиз-кандидат, чтобы собрать отзывы пользователей перед выходом релиза.

В течение периода поддержки в релизной серии выпускаются обновления для новых версий дистрибутивов Linux.

Срок поддержки для каждой серии — как минимум два года.

### Прекращение поддержки

Стадия прекращения поддержки (end of life, EOL) для серии наступает, когда опубликован последний релиз в этой серии. После этого серия прекращает обновляться.

Для серий, достигших этой стадии, поддержка не гарантируется. Скорее всего, для них не будут выходить обновления, связанные с работой модулей, коннекторов и инструментов.

Стадия прекращения поддержки для релизной серии может наступить только тогда, когда большинство производственных сред, с которыми Tarantool работает по SLA и другим соглашениям, обновится до более новой серии.

### Версии по стадиям жизненного цикла

| Стадия | Типы версий | Примеры |
|---|---|---|
| Ранняя разработка | Альфа, бета, релиз-кандидат | `3.0.0-alpha1`<br>`3.0.0-beta1`<br>`3.0.0-rc1`<br>`3.0.0-dev` |
| Поддержка | Релиз-кандидат, релиз | `3.0.0`<br>`3.0.1-rc1`<br>`3.0.1-dev` |
| Прекращение поддержки | – | – |

#### Пример релизной серии

На стадии ранней разработки последовательность версий в релизной серии может быть такой:

```
3.0.0-alpha1
3.0.0-alpha2
...
3.0.0-alpha7

3.0.0-beta1
...
3.0.0-beta5

3.0.0-rc1
...
3.0.0-rc4

3.0.0 (релиз)
```

Как только выпущен первый релиз, серия переходит на стадию поддержки. Дальнейшие версии могут быть опубликованы в такой последовательности:

```
3.0.0 (релиз новой основной версии)

3.0.1-rc1
...
3.0.1-rc4
3.0.1 (релиз с исправлением ошибок, но без функциональных изменений)

3.1.0-rc1
...
3.1.0-rc6
3.1.0 (релиз с новыми функциональными возможностями и, вероятно,␣
↪дополнительными исправлениями)
```

Со временем стадия поддержки подходит к концу, и релизная серия выходит на стадию прекращения поддержки (end of life, EOL). С этого момента новые версии не выпускаются.

---

**Примечание:** See all currently supported Tarantool versions in *Releases*.

---

### 11.1.7 Compatibility guarantees

Backwards compatibility is guaranteed between all versions in the same *release series*. It is also appreciated but not guaranteed between different release series (major number changes). Pre-releases and releases of one release series are compatible in all senses defined below (any release with any release):

- Pre-releases and releases of consequent series are compatible by data layout, binary protocol, and replication protocol.
- No guarantees are given regarding compatibility between pre-releases/releases of non-consequent release series if the opposite is not stated in the *release notes*.
- No guarantees are given regarding compatibility between alpha/beta versions and between alpha/beta and pre-release/release even within one series.

### Binary data layout

Any newer release (its runtime) is backward compatible with any older one. It means the more recent release can work on top of data (`*.xlog`, `*.snap`, `*.vylog`, `*.run`) from the older one. All functionality of the older release can work in this configuration. The same compatibility is maintained between *release series* as well.

An attempt to use a new feature results in one of the options:

- The attempt is successful.
- There is an error message about the old data layout. The error does not lead to service outage or data corruption. There is a way to avoid the message, if an instance upgrades the data layout by calling the *box.schema.upgrade()*. The call enables all new release features (when all instances of the replicaset are processed on the same Tarantool version).

### Binary protocol

All binary protocol requests operational in an older release keep working in a newer one. Responses have the same format, but mappings may contain fields not present in the older release.

A `net.box` client of an older release can work with a server running a newer release. However, `net.box` features introduced in the newer release won't work. A `net.box` client of a newer release is fully operational with a server running a older release. However, only the features implemented in the older release will work.

### Replication protocol

An instance running on a newer release can work as:

- upstream (master) of an instance with an older release
- downstream (replica) without database schema upgrade.

The database schema upgrade (`box.schema.upgrade()`) must be performed when all replicaset instances run on the same Tarantool version. An application should not lean on internal schema representation because it can be changed with the upgrade.

### Lua code

If a code is processed on an older release, it will operate with the same effect on a newer one. However, only meaningful code counts. If any code throws an error but starts doing something useful, the change is considered compatible.

There is still room for new functionality: adding new options (fields in a table argument), new arguments to the end, more fields to a return table, and more return values (multireturn).

Adding a new built-in module or a new global value is considered as a compatible change.

Adding a new field to an existing metatable is okay if the field is not listed in the Lua 5.1 Reference Manual. Otherwise, it should be proven that it won't break any meaningful code.

Examples of compatible changes:

- Add `__pairs`, `__ipairs` to a metatable of a userdata/cdata object. The fields are not from Lua 5.1, and the userdata/cdata has no default behaviour for `pairs()` and `ipairs()` calls.
- Add or extend the `__lt` or `__le` metamethod (if the attempt to use `<`, `<=` etc. leads to an error before the change).

- Extend existing `__eq` metamethod implementation (if the attempt to use it leads to an error before the change).

Examples of **incompatible** changes:

- Add `__pairs`, `__ipairs` to a metatable of a table (it already has a defined behavior before the change).
- Add the `__eq` metamethod (any pair of Lua objects already has a defined behavior).

### SQL code

If any request is processed on an older release, it will operate with the same effect on a newer one (except the requests that always lead to an error).

Examples of compatible changes:

- Add a new keyword.
- Add a new type.
- Add a new built-in function.
- Add a new system table that has a name starting with an underscore.
- Add a new collation.
- Add an implicit or explicit cast rule for a set of operations {X} and a list of types [Y] if [operation from {X}]([list of values of [Y] types]) had not been implemented before the change.
- Change the order of tuples in the result set of `SELECT` in case `ORDER BY` is not specified.

Technically, those changes may break some working code in case of a name clash, but the probability of it is negligible.

Examples of **incompatible** changes:

- Change the result of working implicit or explicit cast.
- Change of a literal type.

### C code

If a module or a C stored procedure runs on an older release, it will operate with the same effect on a newer one.

It is okay to add a new function or structure to the public C API. It must use one of the Tarantool prefixes (`box_`, `fiber_`, `luaT_`, `luaM_` and so on) or some new prefix.

A symbol from a used library must not be exported directly because the library may be used in a module by itself, and the clash can lead to problems. Exception: when the whole public API of the library is exported (as for libcurl).

Do not introduce new functions or structures with the `lua_` and `luaL_` prefixes. Those prefixes are for the Lua runtime. Use `luaT_` for Tarantool-specific functions, and `luaM_` for general-purpose ones.