RethinkDB

faq docs api community blog

The open-source database
for the realtime web

Install RethinkDB
>

Latest release: RethinkDB 2.4.2 (Night Of The Living Dead) 23k stars on GitHub @rethinkdb on Twitter

Official drivers

JS JavaScript

Ruby

Python

Java

Current community-supported drivers

These drivers have been updated to use the JSON driver protocol and at least RethinkDB 2.0 ReQL.

C# C# bchavez

C# C# mfenniak

C++ C++

Clojure

CL Common Lisp

Dart

Data Explorer

History

```
1 var r = require("rethinkdb");
2 r.db('test').tableCreate('tv_shows');
3 r.table('tv_shows').insert([{ name: 'Star Trek TNG', episodes: 178 },
4                             { name: 'Battlestar Galactica', episodes: 75 }])
```

Clear Run

1 row returned in 3ms.

Tree view | Table Viewer View | Table view | Raw view | Query profile


```
{
  "deleted": 0 ,
  "errors": 0 ,
  "generated_keys": [
     "f7c927d5-7bc8-48c2-9bb5-039382049aaa" ,
     "fbeb1363-2a63-46a8-bfdb-d1ba2ccdc0e3"
  ] ,
  "inserted": 2 ,
  "replaced": 0 ,
  "skipped": 0 ,
  "unchanged": 0
}
```

Data Explorer

History

```
1 r.table('tv_shows').count()
```

Clear Run

1 row returned in 1ms.

Tree view | Table Viewer View | Table view | Raw view | Query profile

2

Data Explorer

History

```
1 r.table('tv_shows').filter(r.row('episodes').gt(100))
```

Clear Run

1 row returned.

Tree view | Table Viewer View | Table view | Raw view | Query profile


```
{
  "episodes": 178 ,
  "id": "f7c927d5-7bc8-48c2-9bb5-039382049aaa" ,
  "name": "Star Trek TNG"
}
```

Data Explorer

History

```
r.db('test').tableCreate('users')
r.table('users').insert([{
  id: 10001,
  name: "Bob Smith",
  contact: {
    phone: {
      work: "408-555-1212",
      home: "408-555-1213",
      cell: "408-555-1214"
    },
    email: {
      work: "bob@smith.com",
      home: "bobsmith@gmail.com",
      other: "bobbys@moosecall.net"
    },
    im: {
      skype: "Bob Smith",
      aim: "bobmoose",
      icq: "nobodyremembersicqnumbers"
    }
  },
  notes: [
    {
      date: r.time(2014,1,1,'Z'),
      from: "John Doe",
      subject: "My name is even more boring than Bob's"
    },
    {
      date: r.time(2014,2,2,'Z'),
      from: "Bob Smith Sr",
      subject: "Happy Second of February"
    }
  ]
}])
```

Data Explorer

History

```
1 r.table('users').get(10001)('contact')('phone')('work')
```

Clear Run

1 row returned in <1ms.

Tree view | Table Viewer View | Table view | Raw view | Query profile

```
"408-555-1212"
```

Data Explorer

History

```
r.db('test').tableCreate('reviews');
r.table('reviews').insert([{ name: 'Star Trek TNG', score: 10 },
                           { name: 'Star Trek TNG', score: 9 }]);
```

Clear Run

1 row returned in 10ms.

Tree view | Table Viewer View | Table view | Raw view | Query profile


```
{
  "deleted": 0 ,
  "errors": 0 ,
  "generated_keys": [
     "07ed3575-03f0-464d-ba73-314eaa282a3d" ,
     "b058896f-e6b2-40e6-aa32-120c581300fc"
  ] ,
  "inserted": 2 ,
  "replaced": 0 ,
  "skipped": 0 ,
  "unchanged": 0

}
```

Data Explorer

History

```
1 r.table("reviews").indexCreate("name")
```

Clear Run

1 row returned in 10ms.

Tree view | Table Viewer View | Table view | Raw view | Query profile


```
{
  "created": 1
}
```

Data Explorer

History

```
1 r.table("tv_shows").eqJoin("name", r.table("reviews"), {index: "name"})
```

Clear Run

2 rows returned. Displaying rows 0-1

Tree view | Table Viewer View | Table view | Raw view | Query profile


```
{
  "left": {
    "episodes": 178 ,
    "id": "f7c927d5-7bc8-48c2-9bb5-039382049aaa" ,
    "name": "Star Trek TNG"
  } ,
  "right": {
    "id": "b058896f-e6b2-40e6-aa32-120c581300fc" ,
    "name": "Star Trek TNG" ,
    "score": 9
  }
}
{
  "left": {
    "episodes": 178 ,
    "id": "f7c927d5-7bc8-48c2-9bb5-039382049aaa" ,
    "name": "Star Trek TNG"
  } ,
  "right": {
    "id": "07ed3575-03f0-464d-ba73-314eaa282a3d" ,
    "name": "Star Trek TNG" ,
    "score": 10
  }
}
```

Contributors 139

+ 128 contributors

Languages

- C++ 80.4%
- Python 17.6%
- JavaScript 0.5%
- C 0.4%
- Ruby 0.3%
- Makefile 0.3%
- Other 0.5%

Creation

```
// Create a secondary index on the last_name attribute
r.table("users").indexCreate("last_name").run(conn,
callback)
```

```
// Wait for the index to be ready to use
r.table("users").indexWait("last_name").run(conn, callback)
```

Querying

```
// Get all users whose last name is "Smith"
r.table("users").getAll("Smith", {index: "last_name"}).run(conn, callback)

// Get all users whose last names are "Smith" or "Lewis"
r.table("users").getAll("Smith", "Lewis", {index: "last_name"}).run(conn, callback)

// Get all users whose last names are between "Smith" and "Wade"
r.table("users").between("Smith", "Wade", {index: "last_name"}).run(conn, callback)

// Efficiently order users by last name using an index
r.table("users").orderBy({index: "last_name"}).run(conn, callback)

// For each blog post, return the post and its author using the last_name index
// (assume "author_full_name" is the name of a field in "posts")
r.table("posts").eqJoin("author_last_name", r.table("users"), {index: "last_name"}) \
    .zip().run(conn, callback)
```

Creation

```
// Create a compound secondary index based on the first_name and last_name attributes
r.table("users").indexCreate(
    "full_name", [r.row("last_name"), r.row("first_name")]
).run(conn, callback)
```

```
// Wait for the index to be ready to use
r.table("users").indexWait("full_name").run(conn, callback)
```

Querying

```
// Get all users whose full name is John Smith.
r.table("users").getAll(["Smith", "John"], {index: "full_name"}).run(conn, callback)

// Get all users whose full name is between "John Smith" and "Wade Welles"
r.table("users").between(
    ["Smith", "John"], ["Welles", "Wade"], {index: "full_name"}
).run(conn, callback)

// Get all users whose last name is Smith.
r.table("users").between(
    ["Smith", r.minval], ["Smith", r.maxval], {index: "full_name"}
).run(conn, callback)

// Efficiently order users by first name and last name using an index
r.table("users").orderBy({index: "full_name"}).run(conn, callback)

// For each blog post, return the post and its author using the full_name index
r.table("posts").eqJoin(
    "author_full_name", r.table("users"), {index: "full_name"}
).run(conn, callback)
```

```
{
    title: "...",
    content: "...",
    tags: [ <tag1>, <tag2>, ... ]
}
```

```
// Create the multi index based on the field tags
r.table("posts").indexCreate("tags", {multi: true})

// Wait for the index to be ready to use
r.table("posts").indexWait("tags").run(conn, callback)
```

Querying

```
// Get all posts with the tag "travel" (where the field tags contains "travel")
r.table("posts").getAll("travel", {index: "tags"}).run(conn, callback)

// For each tag, return the tag and the posts that have such tag
r.table("tags").eqJoin("tag", r.table("posts"), {index: "tags"}).run(conn, callback)
```

```
} ,
{
    "description": "Perform read." ,
    "duration(ms)": 0.801016 ,
    "sub_tasks": [
        {
            "parallel_tasks": [
                [
                    {
                        "description": "Perform read on shard." ,
                        "duration(ms)": 0.045636 ,
                        "sub_tasks": [
                            {
                                "description": "Do range scan on secondary index." ,
                                "duration(ms)": 0.009266 ,
                                "sub_tasks": [ ]
                            }
                        ]
                    }
                ] ,
                [
                    {
                        "description": "Perform read on shard." ,
                        "duration(ms)": 0.050276 ,
                        "sub_tasks": [
                            {
                                "description": "Do range scan on secondary index." ,
                                "duration(ms)": 0.024168 ,
                                "sub_tasks": [ ]
                            }
                        ]
                    }
                ] ,
                [
                    {
                        "description": "Perform read on shard."
```

Connected to
MacBook_Pro_...

Issues
No issues

Servers
3 connected

Tables
0/0 ready

Servers connected to the cluster

MacBook_Pro_Zogov_local_q6j	default	0 primaries, 0 secondaries	**MacBook-Pro-Zogov.local**, up for a minute
MacBook_Pro_Zogov_local_ekt	default	0 primaries, 0 secondaries	**MacBook-Pro-Zogov.local**, up for a few seconds
MacBook_Pro_Zogov_local_n23	default	0 primaries, 0 secondaries	**MacBook-Pro-Zogov.local**, up for a few seconds

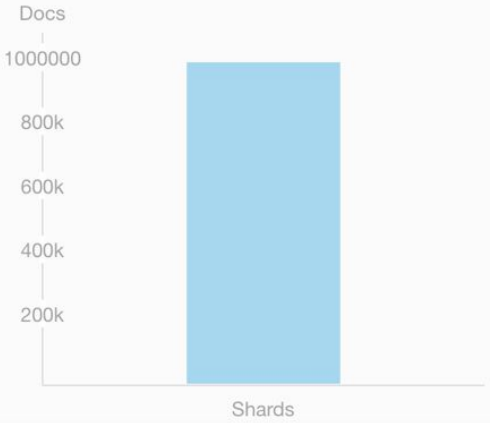
Ready

About 1.0M documents

1/1 primary replicas

1/1 replicas available

Data distribution



Sharding and replication

1 shard 1 replica per shard Reconfigure

Secondary indexes

- No secondary indexes found.

Create a new secondary index →

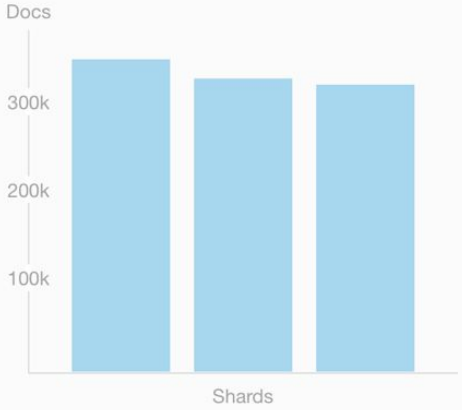
Ready

About 1.0M documents

3/3 primary replicas

6/6 replicas available

Data distribution



Sharding and replication

3 shards 2 replicas per shard Reconfigure

Secondary indexes

- No secondary indexes found.

Create a new secondary index →

Data Explorer

History

```
1 r.table('users').filter(r.row('info.age').gt(40)).filter(r.row('info.age').lt(42)).count()
```

Clear Run

1 row returned in 7.83s.

Tree view | Table Viewer View | Table view | Raw view | Query profile

7.83s round-trip time

7.83s server time

24 shard accesses

Data Explorer

History

```
r.table('users').indexCreate('age', r.row("info")("age"));
r.table('users').between(40, 41, {index: 'age'}).count()
```

Clear Run

1 row returned in 13ms.

Tree view | Table Viewer View | Table view | Raw view | Query profile

18ms round-trip time

13ms server time

24 shard accesses