RethinkDB

25358684

Basic Characteristics

- Document database
- Distributed and easy to scale
- Operations:
  - ReQL = RethinkDB query language
  - MapReduce support
- New access model:
  - instead of polling for changes, RethinkDB can push updated query results to application in real time
- Use cases:
  - realtime web applications
  - e.g. collaborative apps, multiplayer games, connected devices
- Open-source

Install The Server

- Official packages
- Docker image

Run The Server

- run this command in your terminal:
  - `$ rethinkdb`
- open `localhost:8080` in your browser to see the web interface
- or install driver for your favourite language

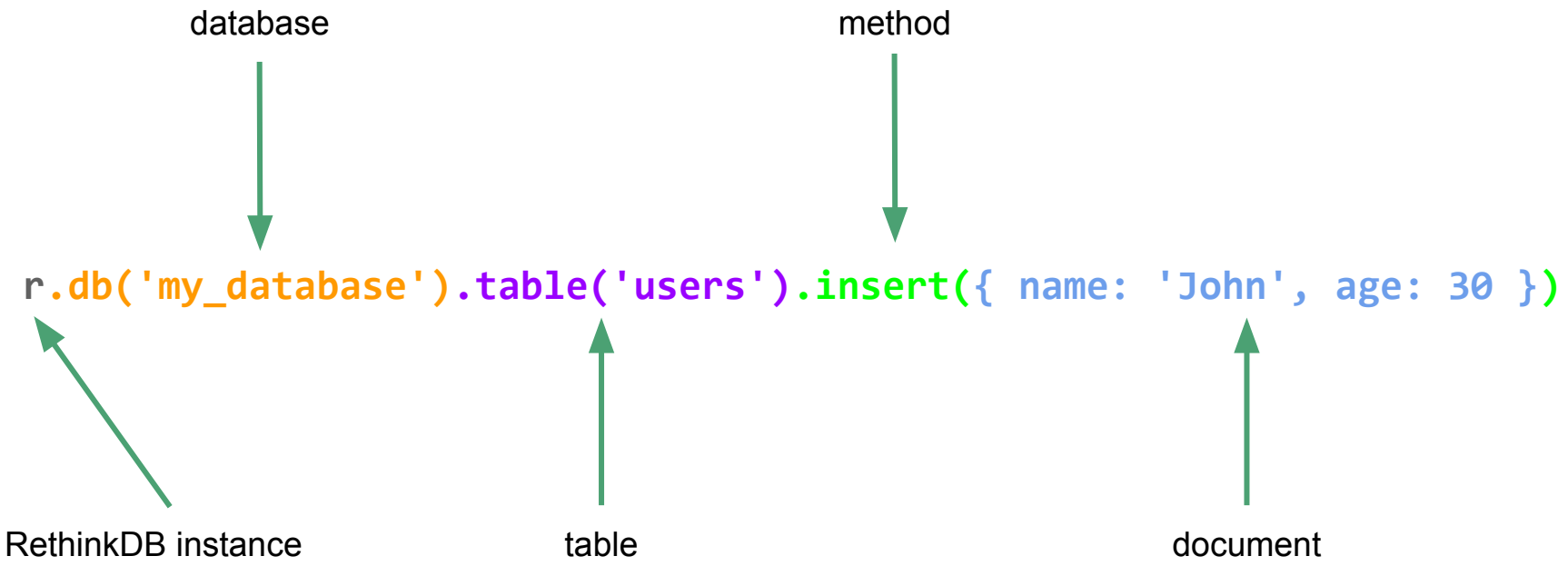
Terminology

SQL	**RethinkDB**
database	database
table	table
row	document
column	field
table joins	table joins
primary key	primary key (by default **`id`**)
index	index

Data Model

- Document = JSON object
  - schemaless
  - unique immutable identifier, field **`id`** by default
- Supported data types
  - basic : number, string, boolean, null, object, array
  - RethinkDB -specific: geometry data types, time objects, streams, ...

ReQL


database
method
r.db('my_database').table('users').insert({ name: 'John', age: 30 })
RethinkDB instance
table
document

DML Operations

```
r.db('my_database').table('users').insert({ name: 'John', age: 30 })
```

- insert new document

```
r.db('my_database').table('users').filter({ name: 'John' }).update({ age: 40 })
```

- update document

```
r.db('my_database').table('users').filter({ name: 'John' }).delete()
```

- delete document

Querying - Simple Select

ReQL

```
r.db('my_database')

.table('users')
```

SQL

```
SELECT * FROM users
```

Querying - Simple Select

ReQL

```
r.db('my_database')
.table('users')
.pluck('name', 'age')
```

SQL

```
SELECT name, age
FROM users
```

Querying - Simple Filter

ReQL

```
r.db('my_database')
.table('users')
.filter({name: 'Peter'})
```

SQL

```
SELECT *
FROM users
WHERE name = 'Peter'
```

Querying - Advanced Filter

ReQL

```
r.db('my_database')
.table('users')
.filter(
      r.row('name').eq('Peter')
      .and(
            r.row('age').lt(30)
      )
)
```

SQL

```
SELECT *
FROM users
WHERE name = 'Peter' AND age < 30
```

Querying - Order By

ReQL

```
r.db('my_database')
.table('users')
.orderBy(
    r.desc('name')
)
```

SQL

```
SELECT *
FROM users
ORDER BY name DESC
```

Querying - Aggregation

ReQL

```
r.db('my_database')
.table('users')
.avg('age')
```

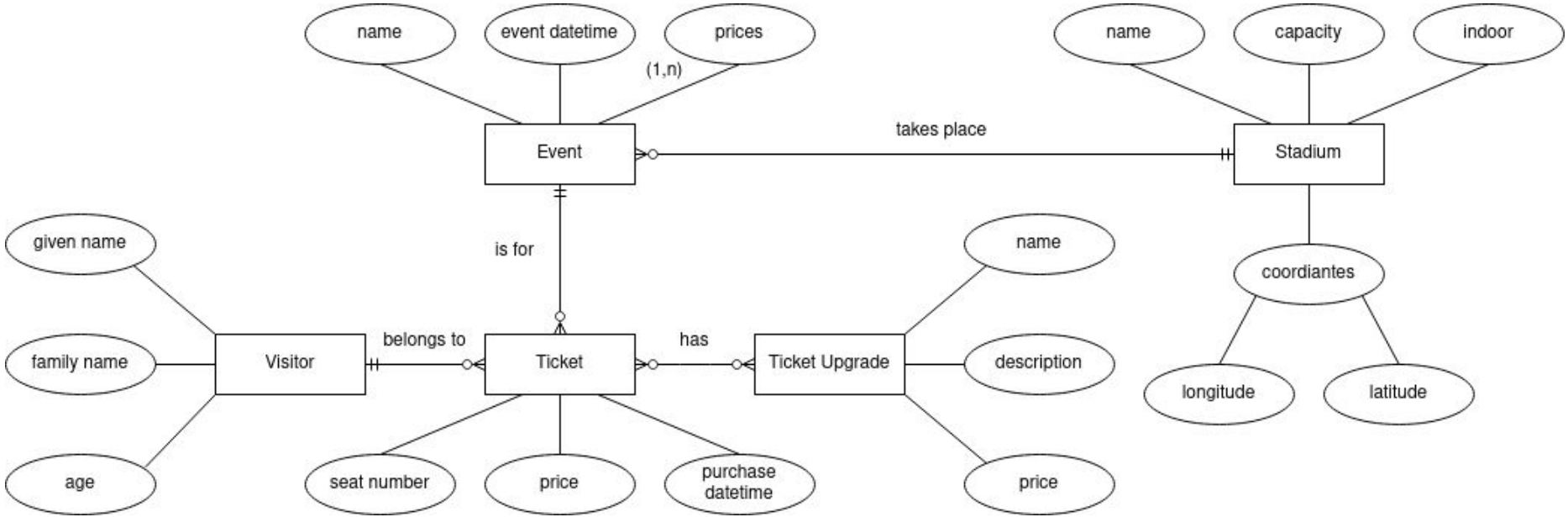
SQL

```
SELECT AVG('age')
FROM users
```

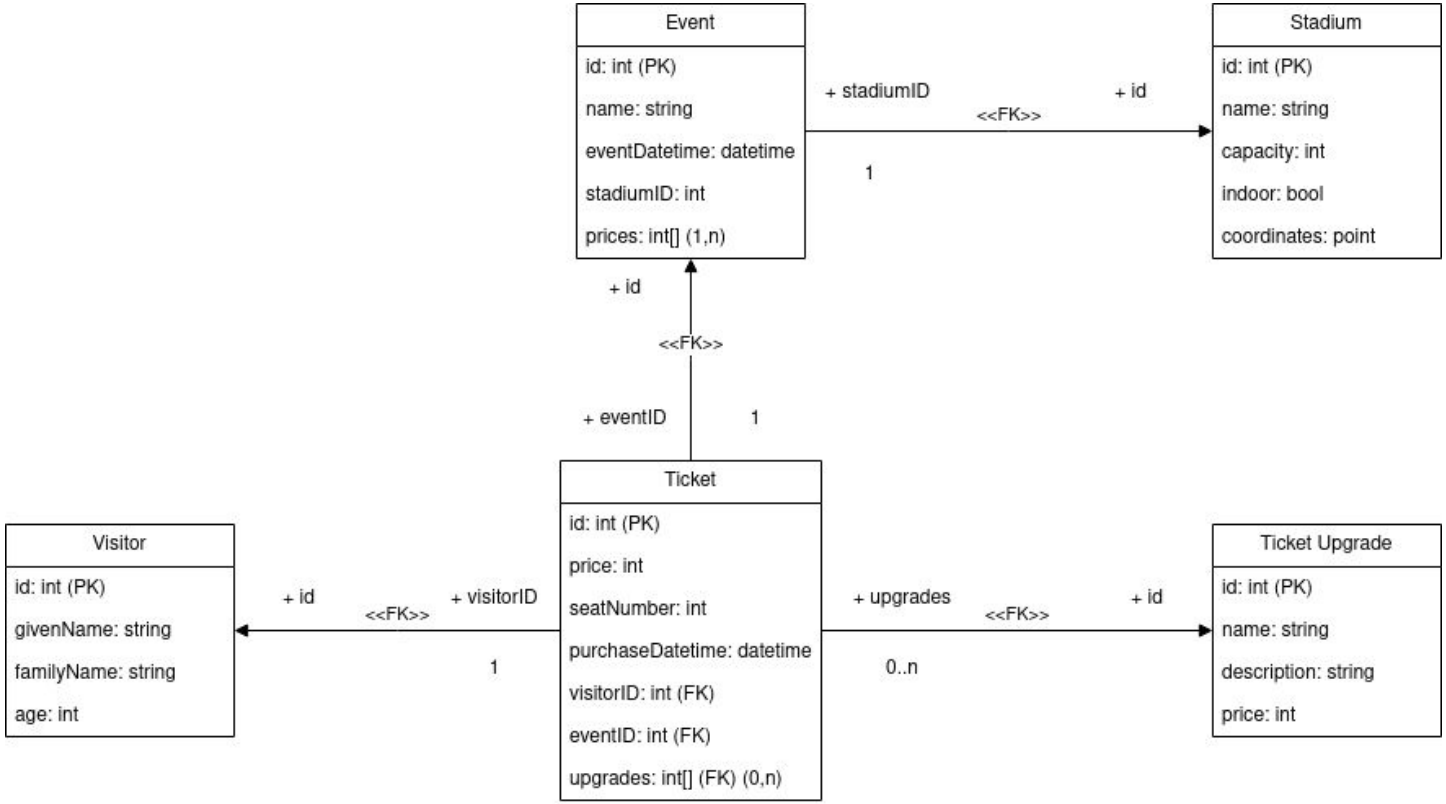
- More examples: SQL to ReQL cheat sheet

Demo: Booking system for sporting events

ER Schema


name
event datetime
prices
(1,n)
Event
takes place
name
capacity
indoor
Stadium
given name
family name
age
Visitor
is for
belongs to
Ticket
has
seat number
price
purchase datetime
name
description
price
Ticket Upgrade
coordiantes
longitude
latitude

Logical Schema



Sample Data

- Sample Data generated and loaded by **`load_data.py`** script
- In numbers:
  - 4 ticket upgrades
  - 40 stadiums
  - 899 visitors
  - 1912 events
  - 85623 tickets

Q1: Attendance In March

- Script **`query_attendance.py`**
- Duration: 58 ms
- Notable parts:
  - multiple nested queries with **`do()`** function

Q2: Stadiums Near Prague

- Script **`query_geospatial.py`**
- Duration: 13 ms
- Geospatial query
- Notable parts:
  - geospatial **`get_nearest()`** function

Q3: Upgraded Tickets

- Script **`query_upgraded_tickets.py`**
- Duration: 848 ms
- Notable parts:
  - aggregation functions **`group()`** and **`count()`**
  - multiple joins

References

- https://rethinkdb.com/