RethinkDB

Niharika Vithala, Deepan Sekar,
Aidan Pace, and Chang Xu

Content

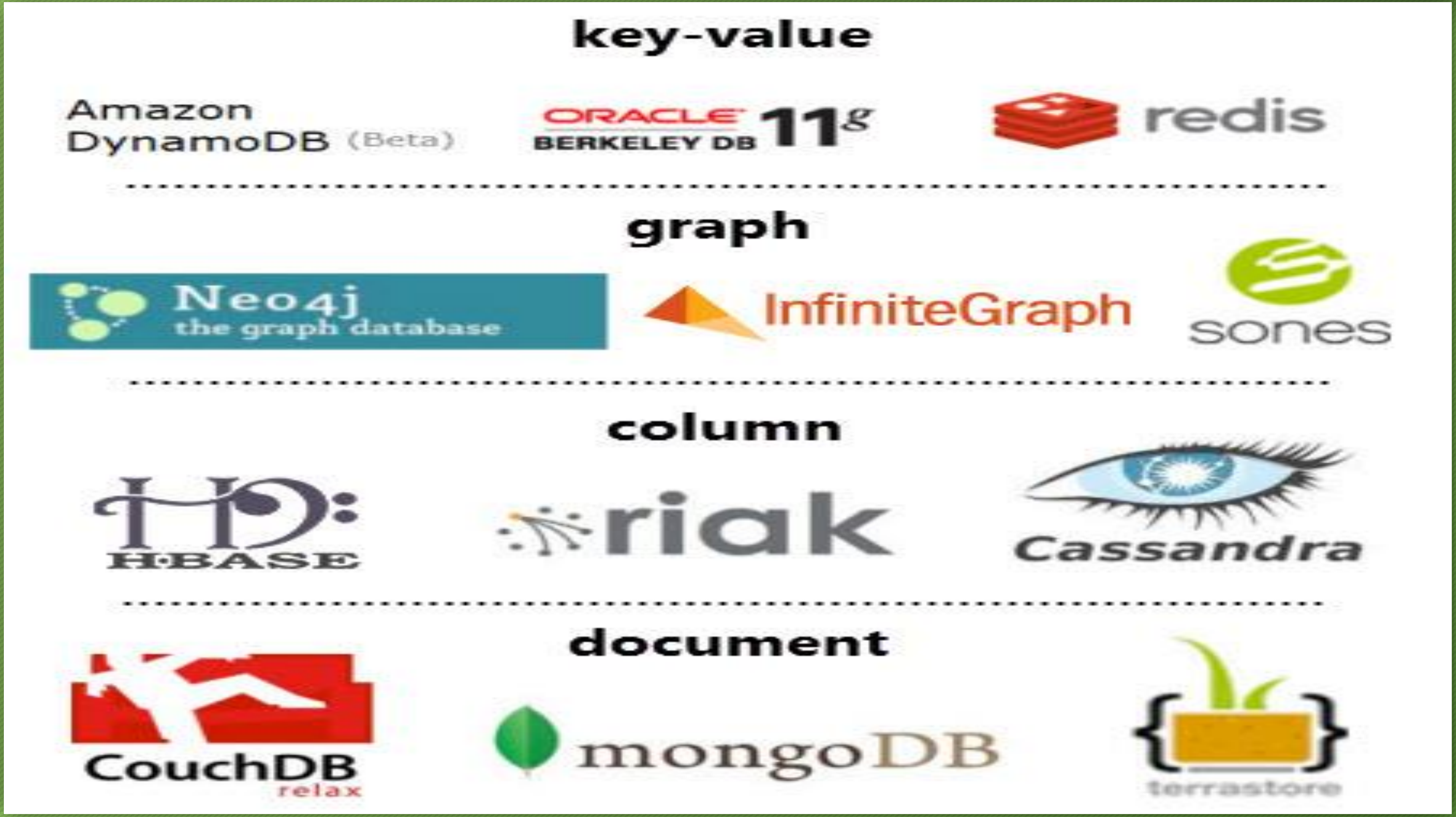
Introduction

Niharika Vithala

What is a NoSQL Database

- Databases that do not necessarily use underlying tabular model for storage and retrieval of data.
- Why did they come into existence?
  - Due to big data applications which generate lot of semi-structured data which often contain valuable information and need to be analyzed and cannot be store in tabular format.
- Some popular examples of NoSql Databases are Cassandra, MongoDB, redis.

Examples


key-value
Amazon DynamoDB (Beta)
ORACLE BERKELEY DB 11g
redis
graph
Neo4j the graph database
InfiniteGraph
sones
column
HBASE
riak
Cassandra
document
CouchDB relax
mongoDB
terrastore

Deciding Between SQL vs NoSQL

- How do we make the call about which database to choose for our underlying application? Some criteria that can help us decide can be as follows:
- We can use NoSql Database when there is huge amount of data which has to be scaled across different nodes for computation.
- When the incoming data to the database does not follow a strict schema. For example, in a social network site where all the data updates have single point of origin which it the user. The application’s interface and performance take a higher priority than robust data integrity. We could use a NoSql data store to implement feature of storing different types of data.

When do we use SQL

- For example in a warehouse application that tracks the arrival of goods into the warehouse, a relational database would serve the need as there is a fixed underlying schema.
- It’s imperative to minimize mistakes. We can’t have products disappearing or being moved to a location where different products are already being stored.
- In its simplest form, we’re recording the transfer of items from one physical area to another — or removing from location A and placing in location B. That’s two updates for the same action.
- We need a robust store with enforced data integrity and transaction support. Only an SQL database will (currently) satisfy those requirements.

What is RethinkDB

- RethinkDB is an open-source, scalable JSON database built specifically for realtime web.
- RethinkDB is designed for utilizations in web and mobile apps, multiplayer games, trading and market and embedded systems. RethinkDB uses JSON format to store and transfer data.
- RethinkDB has been developed completely in C++.
- Instead of polling for changes from the database, the developer can tell RethinkDB to continuously push updated query results to applications in Real-time.
- This feature drastically reduces the amount of time take to build real time applications.

System Features

Niharika Vithala

Push Architecture and Changefeeds

- RethinkDB uses push architecture wherein instead of continuously polling database for changes, changes can automatically be pushed to the application. This is very useful in applications like
  - Collaborative web and mobile apps
  - Streaming analytics apps
  - Multiplayer games
  - Realtime marketplaces
  - Connected devices
- The change feeds architecture is designed to enable each client to open multiple realtime feeds.
- Since modern web and mobile applications often have tens of thousands of concurrent clients, RethinkDB’s feeds are designed to be extremely scalable.
- You should be able to open thousands of concurrent active feeds on a single RethinkDB node, and scale to tens or hundreds of thousands of feeds across a RethinkDB cluster.

Geospatial Queries

- RethinkDB supports spatial and geographic queries through geometry object.
- In RethinkDB, geometry objects are implemented through a geographic coordinate system, with points and shapes plotted on the surface of a sphere in three-dimensional space.
- There are inbuild commands in ReQl that serve this purpose. For example, to get the nearest point to given point.
- point = r.point(-122.422876,37.777128) # San Francisco
- r.table('geo').get_nearest(point, {:index => 'location'})

Data Model

Deepan Sekar

Data types

The `typeOf` command can be appended to any ReQL command to display what data type that command will returns. For instance (in JavaScript):

```
r.table('users').get(1).typeOf().run(conn, callback)
```

Basic data types

- **Numbers**
- **Strings**
- **Booleans**
- **Null**
- **Objects**
- **Arrays**

RethinkDB-specific data types

- **Databases**
- **Tables**
- **Streams**
- **Selections**
- **Pseudo-types**
  - **Binary Objects**
  - **Times**
  - **Geometric types**
  - **Grouped data**

Abstract data types

- **A datum**
- **Sequence**
- **Maxval and Minval**
- **Functions**

Geometry data types

- **Points**
- **Lines**
- **Polygons**

Note:

- **Working with streams**
- **Sorting**

Dates and Times in RethinkDB

- Times are integrated with the official drivers
- Queries are timezone-aware
- Times work as indexes
- Time operations are pure ReQL

Note: Working with Time queries

Binary Objects and Geospatial queries

- RethinkDB supports a native binary object type, letting you use ReQL to store binary objects directly in the database.
- Storing files and User Avatars in the Database.

- Accessing Geo Spatial Locations.
- The Coordinate System

Modeling Relationships

There are two ways to model relationships between documents in RethinkDB:

- By using **embedded arrays**.
- By linking documents stored in **multiple tables** (similar to traditional relational database systems).

Embedded Arrays

Advantages of using embedded arrays:

- Queries for accessing authors and posts tend to be simpler.
- The data is often co-located on disk. If you have a dataset that doesn’t fit into RAM, data is loaded from disk faster.
- Any update to the authors document atomically updates both the author data and the posts data.

Disadvantages of using embedded arrays:

- Deleting, adding or updating a post requires loading the entire `posts` array, modifying it, and writing the entire document back to disk.
- Because of the previous limitation, it’s best to keep the size of the `posts` array to no more than a few hundred documents.

Linking documents in multiple tables

Advantages of using multiple tables:

- Operations on authors and posts don’t require loading the data for every post for a given author into memory.
- There is no limitation on the number of posts, so this approach is more suitable for large amounts of data.

Disadvantages of using multiple tables:

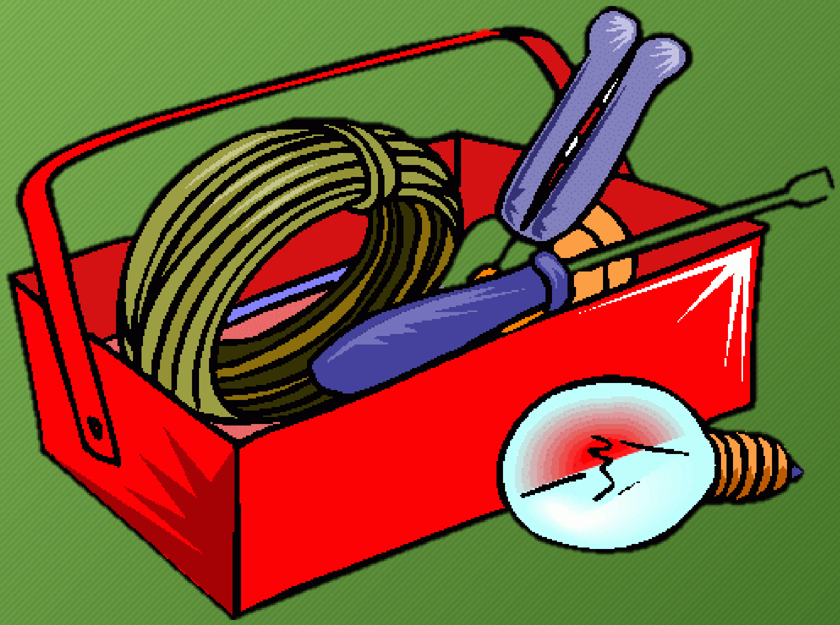
- The queries linking the data between the authors and their posts tend to be more complicated.
- With this approach you cannot atomically update both the author data and the posts data.

ReQL

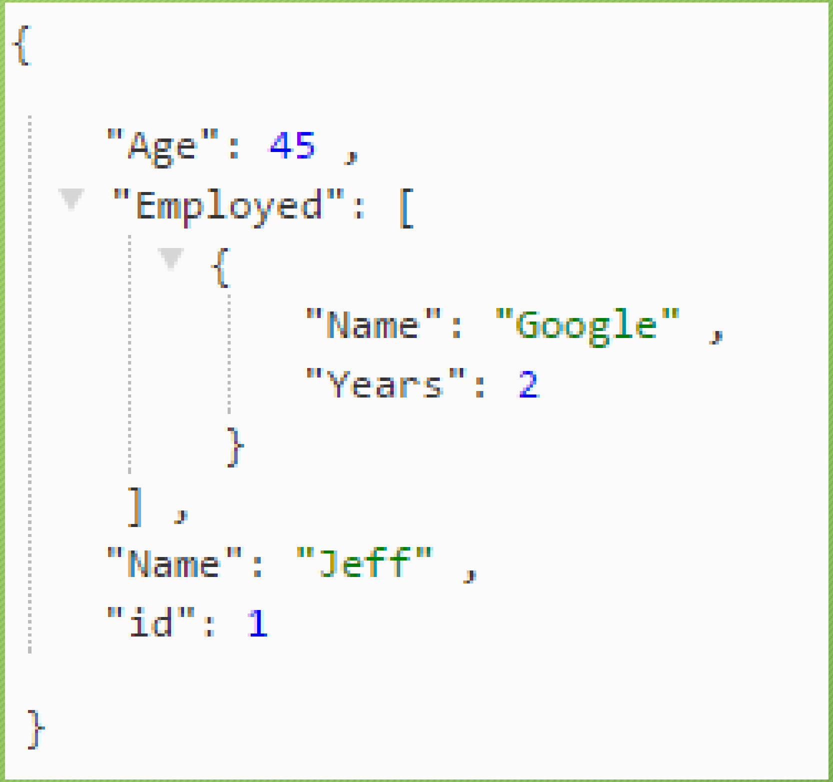
Aidan Pace

NoSQL With a Twist!

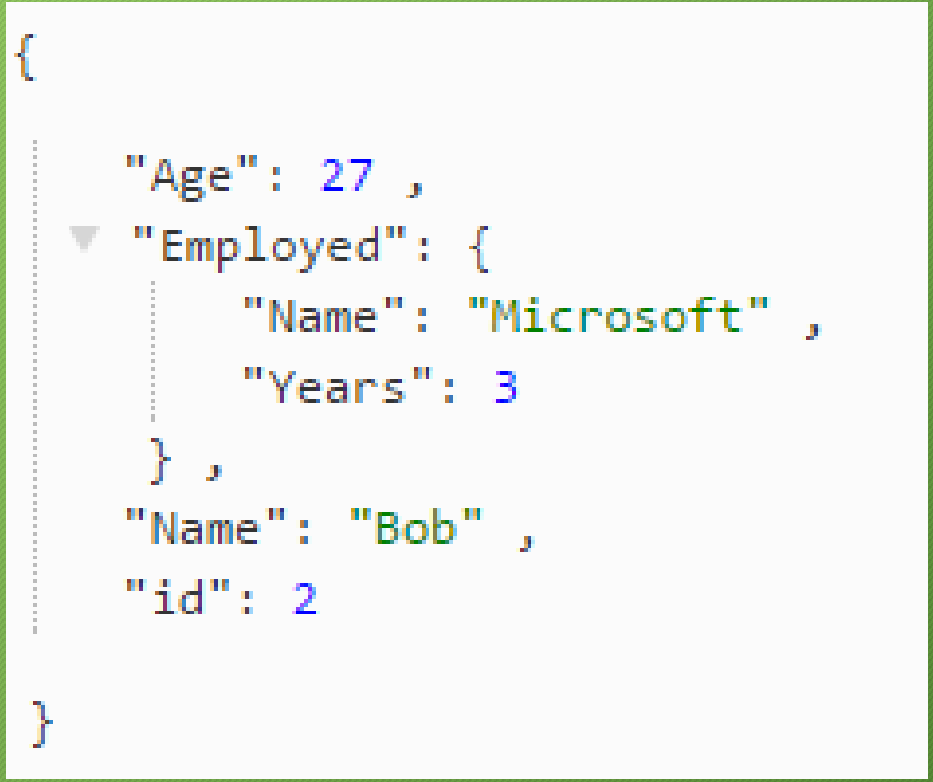
- Embeds into programming language
- Chainable
- Server executed
- Unstructured Data



Operates On JSON Documents



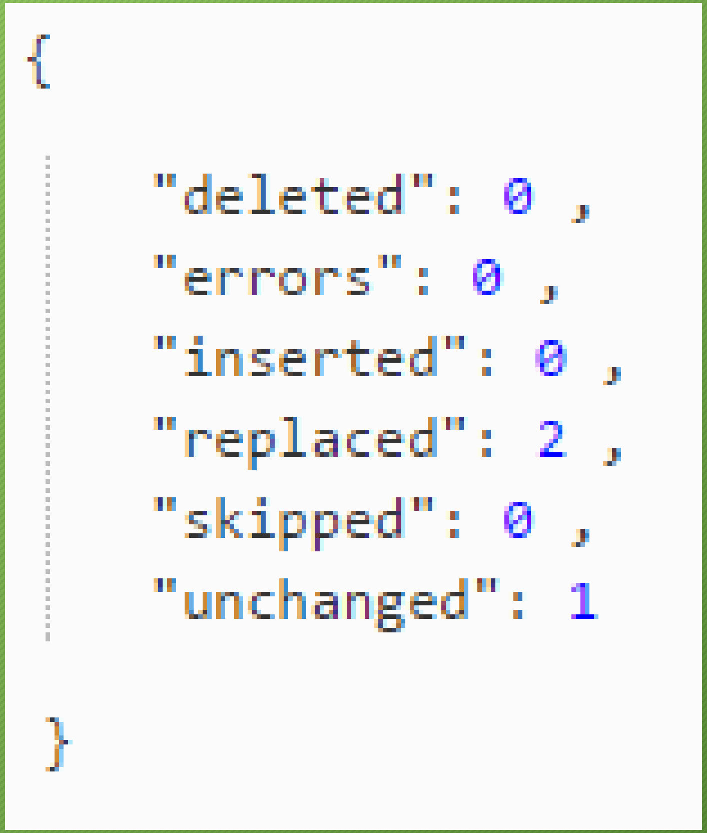
```
{
    "Age": 45 ,
    "Employed": [
        {
            "Name": "Google" ,
            "Years": 2
        }
    ] ,
    "Name": "Jeff" ,
    "id": 1
}
```



```
{
    "Age": 27 ,
    "Employed": {
        "Name": "Microsoft" ,
        "Years": 3
    } ,
    "Name": "Bob" ,
    "id": 2
}
```

Results Are JSON Documents Too

While the results of most queries are very obviously going to be JSON documents, this holds true for all operations, including inserts, deletions, and updates



```
{
    "deleted": 0 ,
    "errors": 0 ,
    "inserted": 0 ,
    "replaced": 2 ,
    "skipped": 0 ,
    "unchanged": 1
}
```

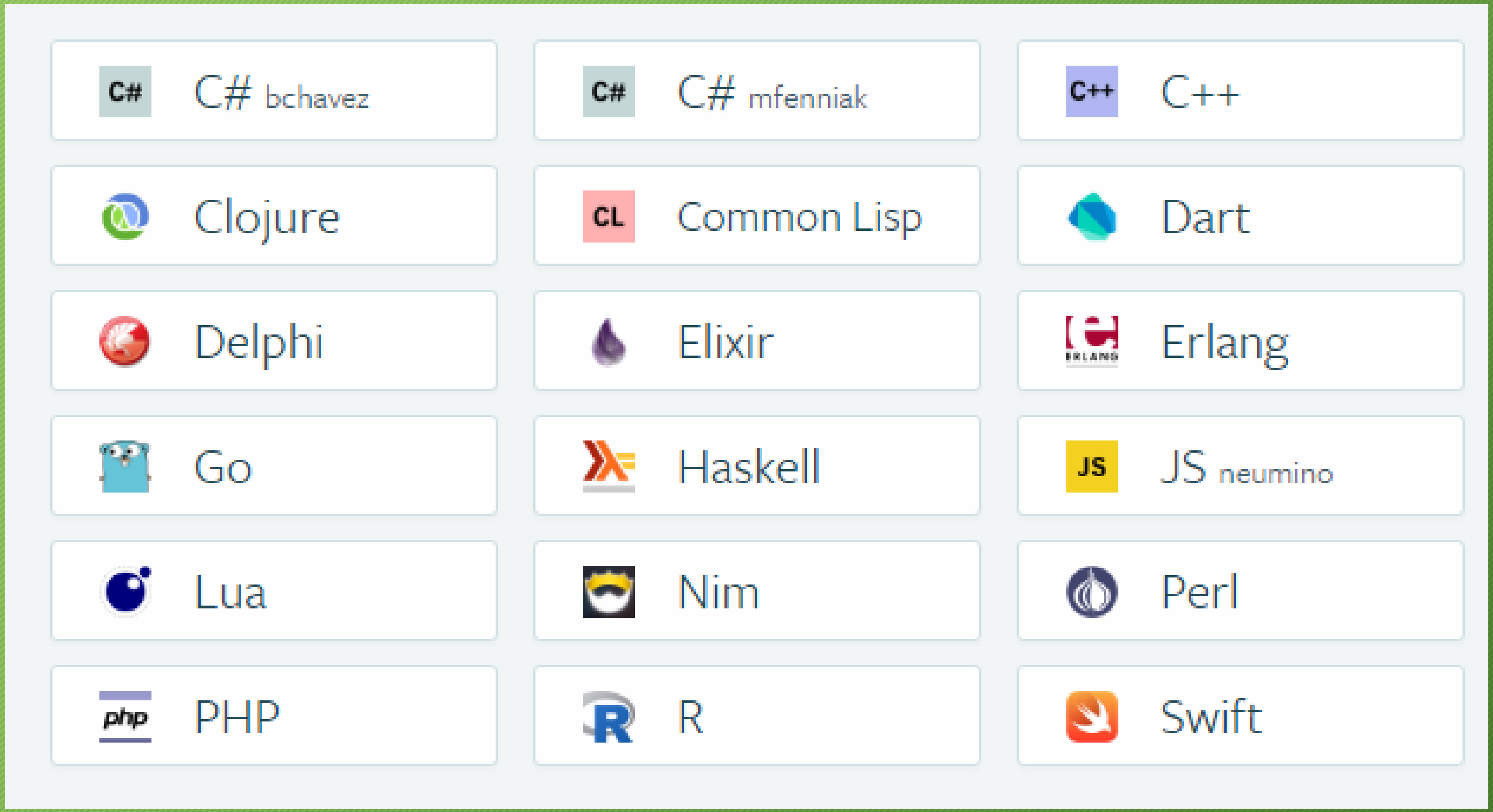
Official Drivers | Community Supported Drivers

JavaScript

Java

RUBY Programming

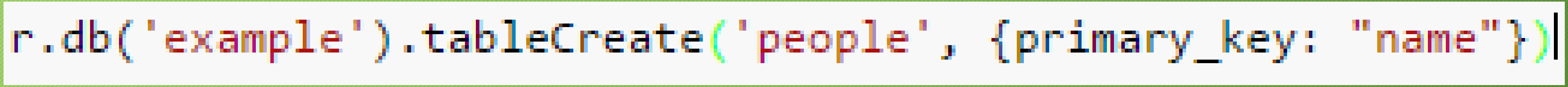
python™



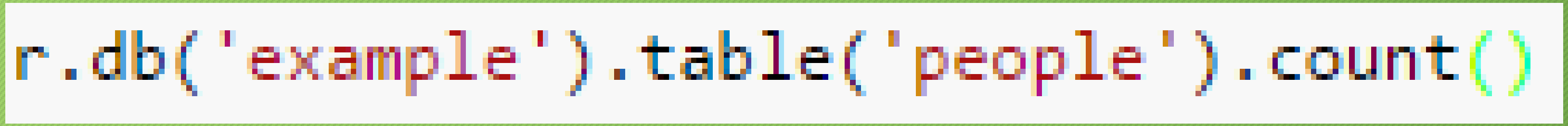
C# bchavez	C# mfenniak	C++
Clojure	Common Lisp	Dart
Delphi	Elixir	Erlang
Go	Haskell	JS neumino
Lua	Nim	Perl
PHP	R	Swift

Chaining

- Examples



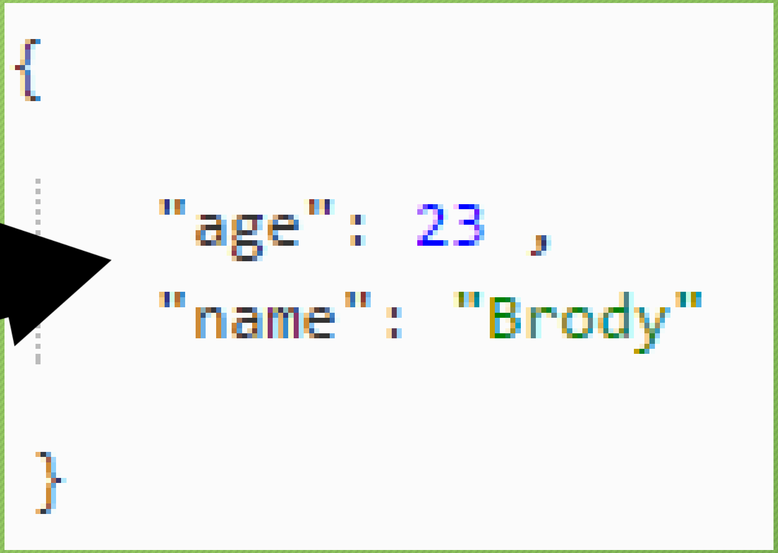
```
r.db('example').tableCreate('people', {primary_key: "name"})
```



```
r.db('example').table('people').count()
```

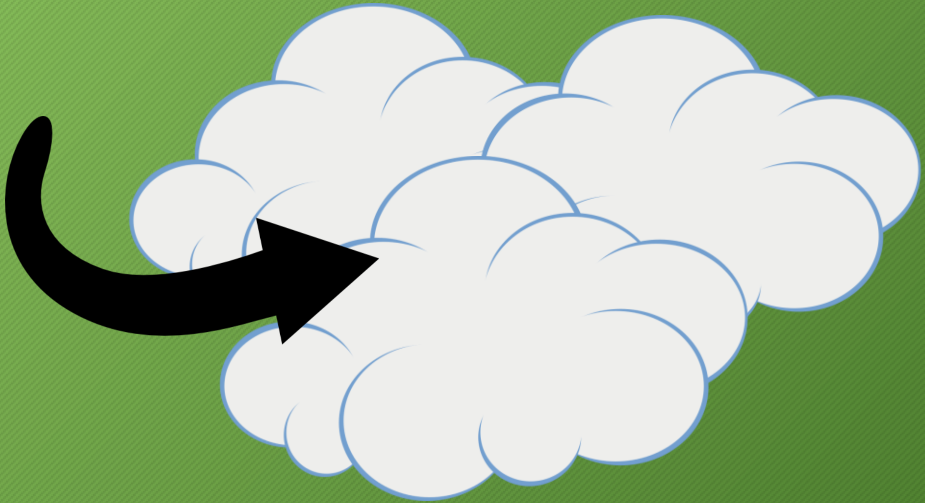
Insertion / Deletion

```
r.db('example').table('people').insert(
  { name: "Brody",
    age: 23 })
```



```
{
    "age": 23 ,
    "name": "Brody"
}
```

```
r.db('example').table('people')
  .get("Brody").delete()
```



Delete or Replace?

- Delete() reserved for documents, tables, and databases
- Replace() used for changing or remove fields in an existing document

Plucking

```
r.db('example').table('people').get('Bobby').pluck('age', 'job')
```

```
{

    "age": 26 ,
    "job": "Programmer" ,
    "name": "Bobby"

}
```

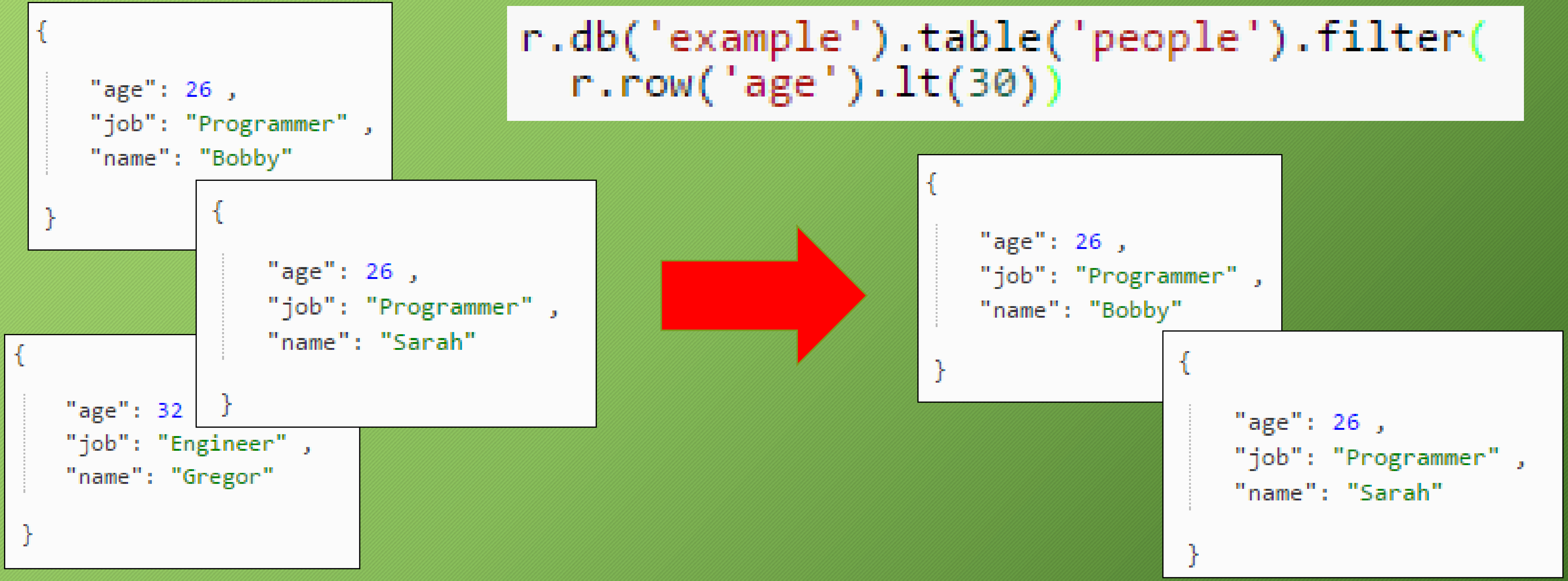
```
{

    "age": 26 ,
    "job": "Programmer"

}
```

Filtering

```
r.db('example').table('people').filter(
  r.row('age').lt(30))
```



```
{
    "age": 26 ,
    "job": "Programmer" ,
    "name": "Bobby"
}
```

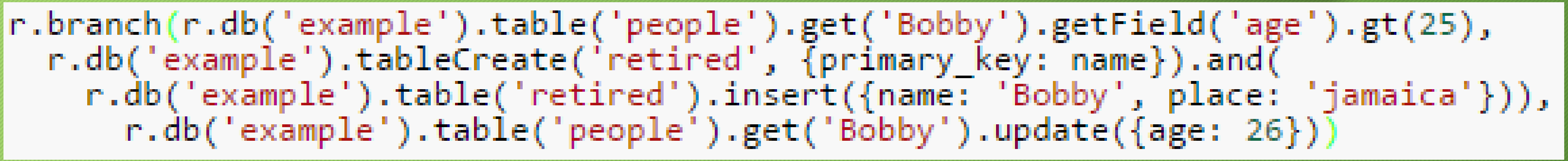
```
{
    "age": 26 ,
    "job": "Programmer" ,
    "name": "Sarah"
}
```

```
{
    "age": 32 ,
    "job": "Engineer" ,
    "name": "Gregor"
}
```

```
{
    "age": 26 ,
    "job": "Programmer" ,
    "name": "Bobby"
}
```

```
{
    "age": 26 ,
    "job": "Programmer" ,
    "name": "Sarah"
}
```

Branch Logic



```
r.branch(r.db('example').table('people').get('Bobby').getField('age').gt(25),
  r.db('example').tableCreate('retired', {primary_key: name}).and(
    r.db('example').table('retired').insert({name: 'Bobby', place: 'jamaica'})),
      r.db('example').table('people').get('Bobby').update({age: 26}))
```

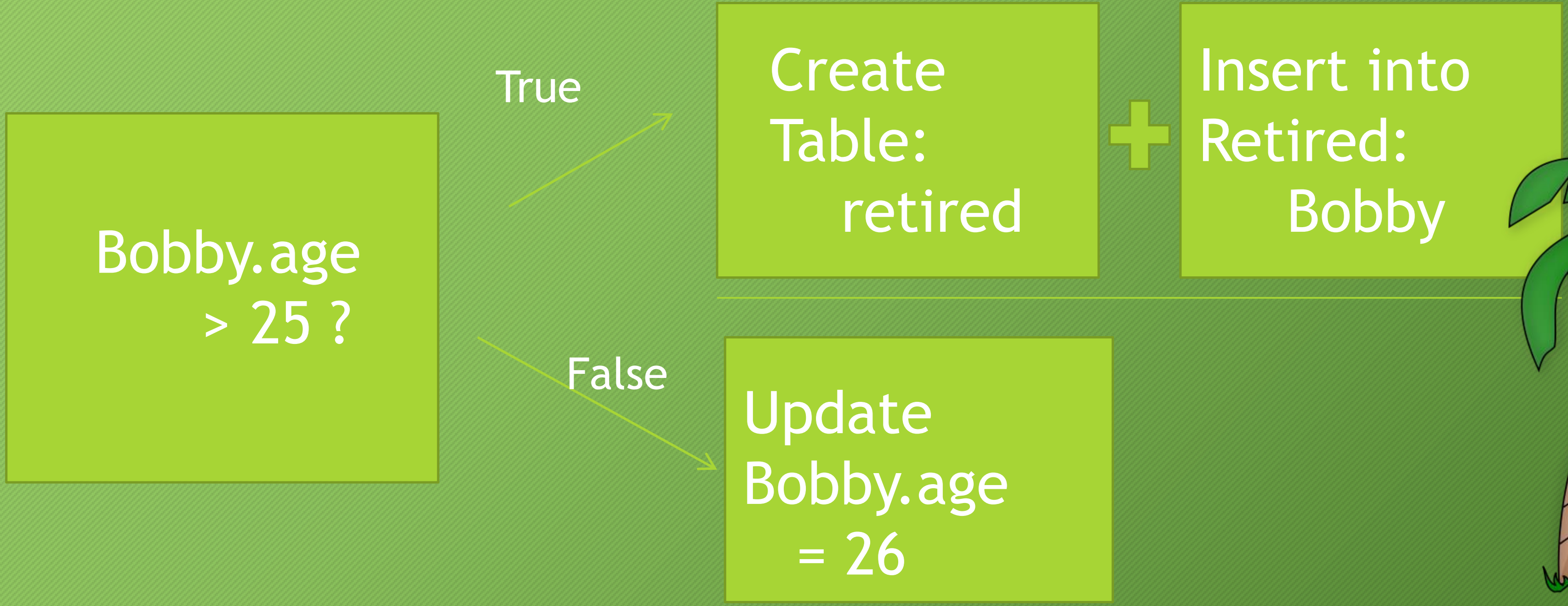
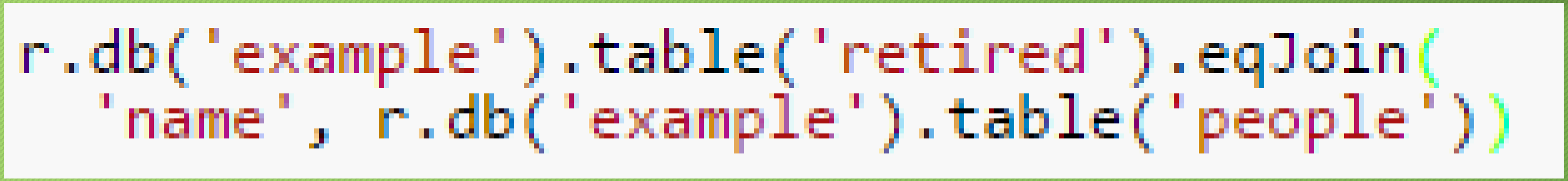
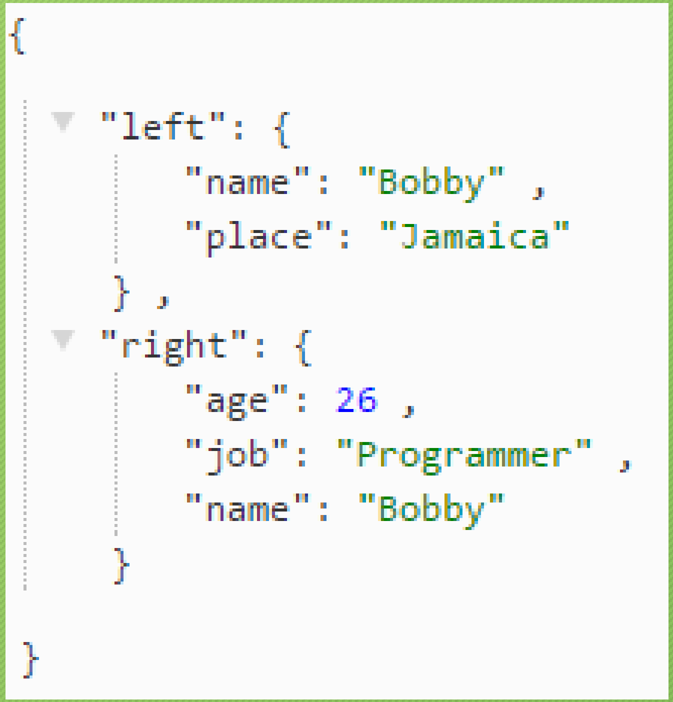


Table Joins

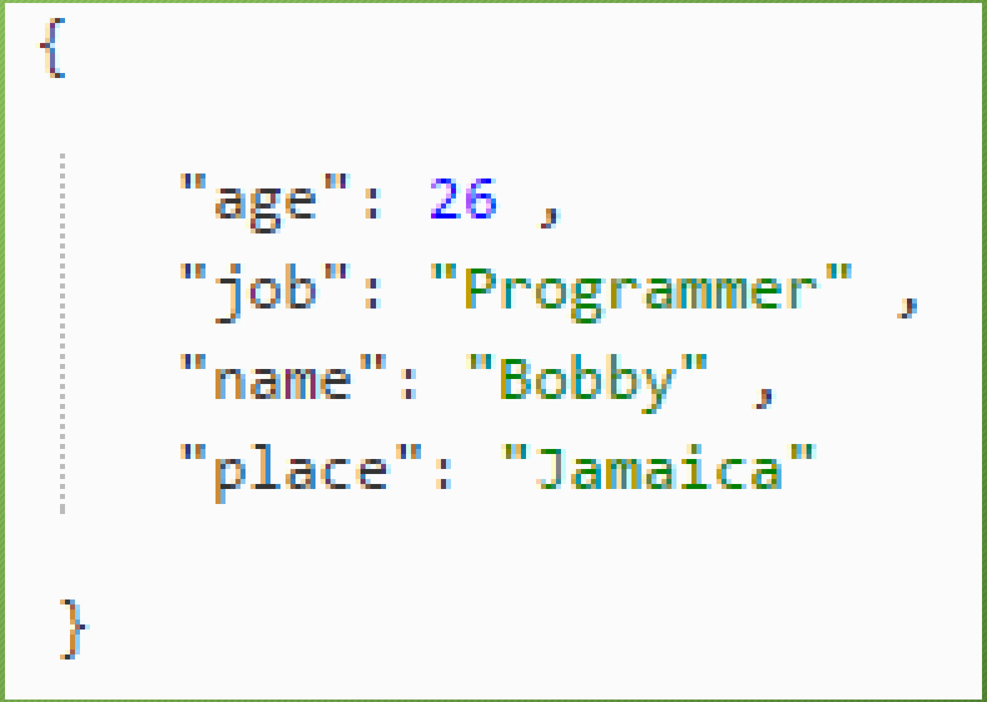


```
r.db('example').table('retired').eqJoin(
  'name', r.db('example').table('people'))
```



```
{
    "left": {
        "name": "Bobby" ,
        "place": "Jamaica"
    } ,
    "right": {
        "age": 26 ,
        "job": "Programmer" ,
        "name": "Bobby"
    }
}
```

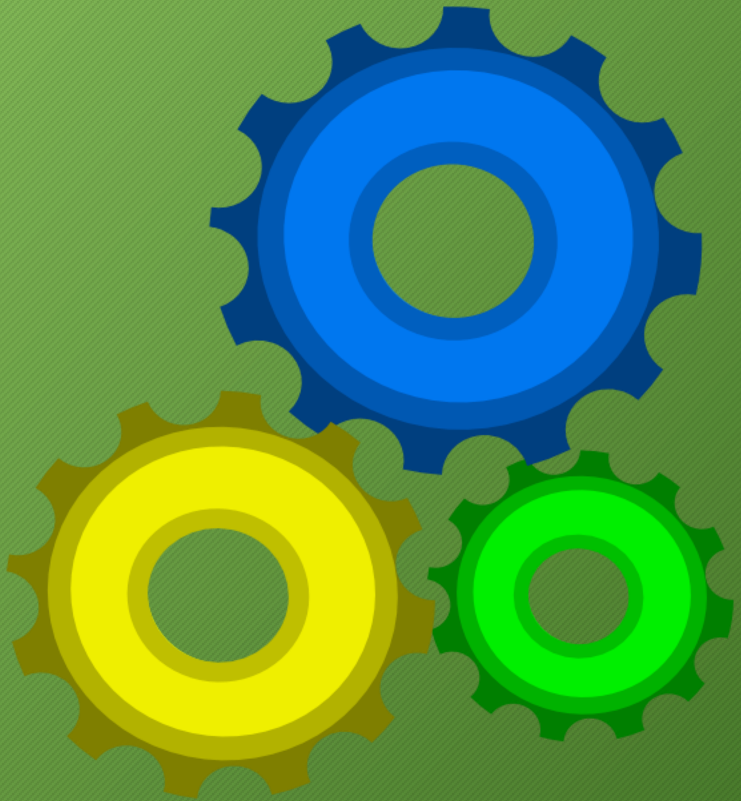
.zip()



```
{
    "age": 26 ,
    "job": "Programmer" ,
    "name": "Bobby" ,
    "place": "Jamaica"
}
```

Aggregate Functions

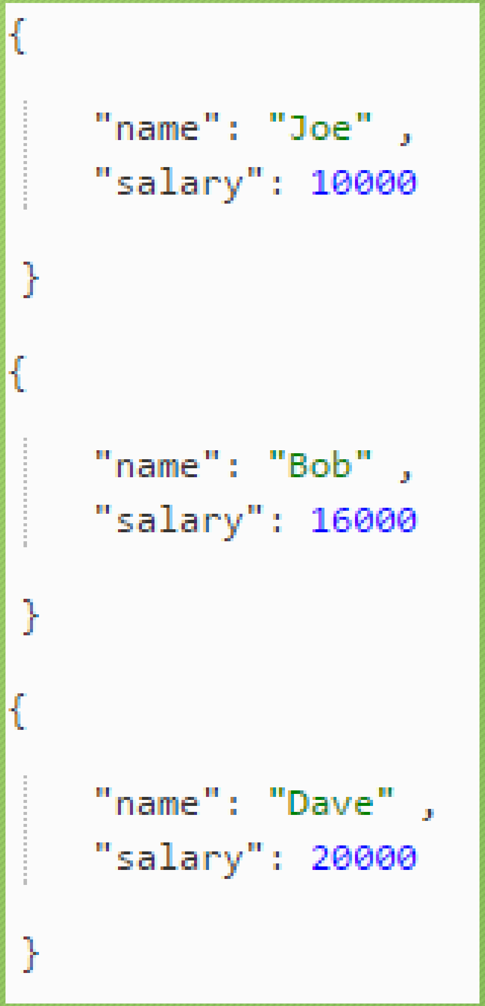
- Group
- Count
- Sum
- Avg
- Min
- Max
- Distinct



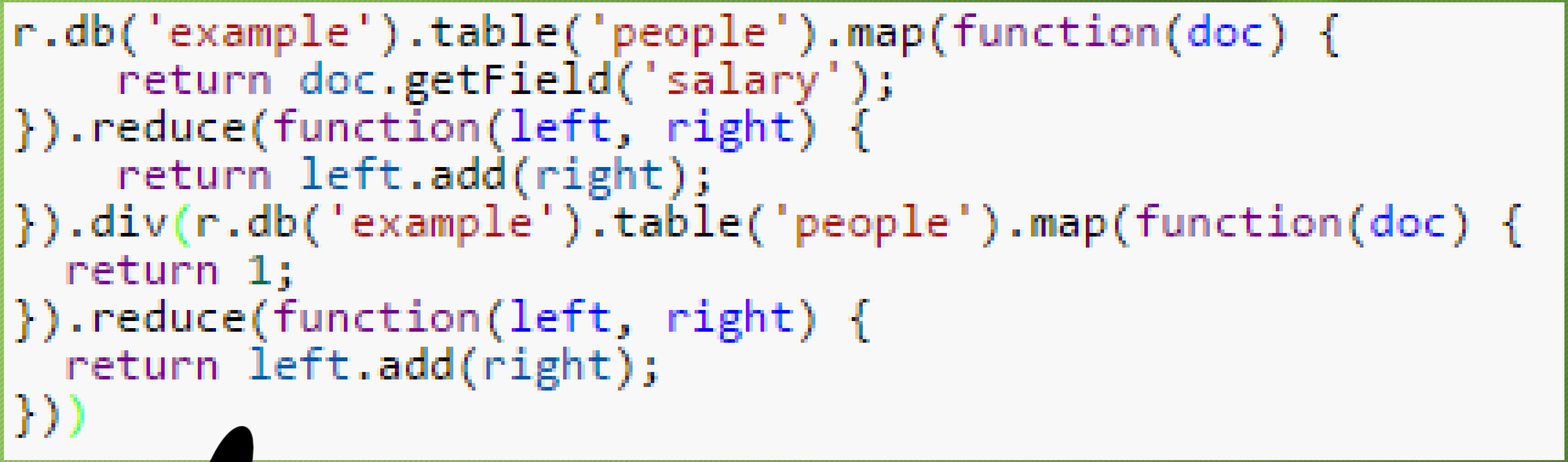
Map / Reduce

- Used to perform operations on large amounts of documents
- Map will transform each element it affects to something else depending on its function
- Reduce will perform an operation on each mapped element and produce a single value from a sequence of operations

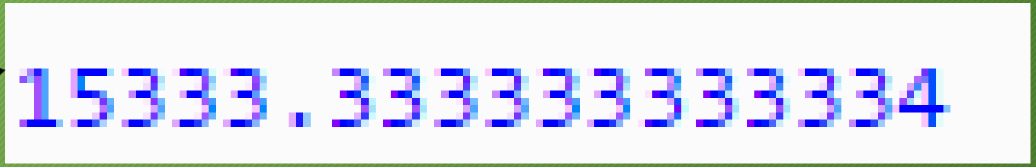
Map / Reduce cont.



```
{
    "name": "Joe" ,
    "salary": 10000
 }
{
    "name": "Bob" ,
    "salary": 16000
 }
{
    "name": "Dave" ,
    "salary": 20000
 }
```



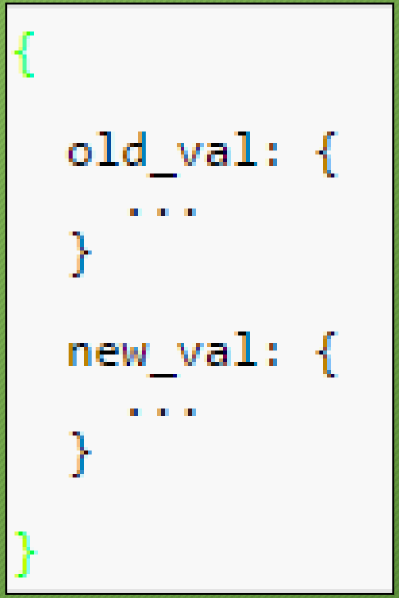
```
r.db('example').table('people').map(function(doc) {
    return doc.getField('salary');
}).reduce(function(left, right) {
    return left.add(right);
}).div(r.db('example').table('people').map(function(doc) {
  return 1;
}).reduce(function(left, right) {
  return left.add(right);
}))
```



```
15333.333333333334
```

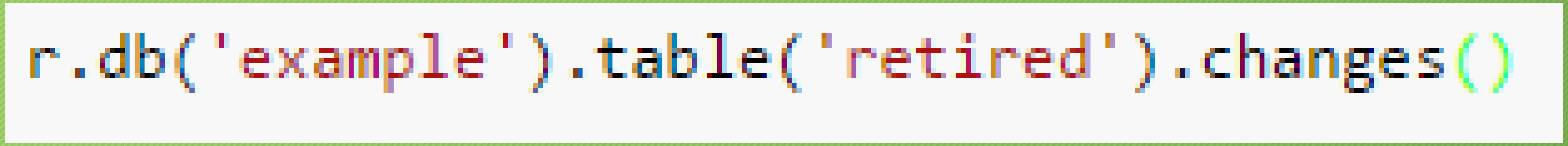
Changefeeds?

- Push model vs. pull model
- Allows subscriptions to elements with the changes() function
- Will push results that have the form:

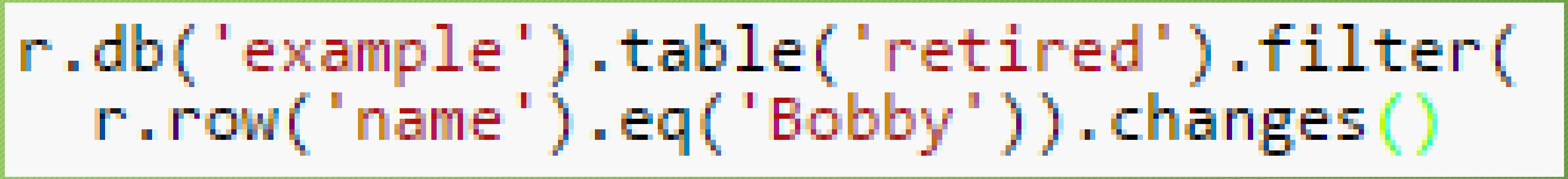


```
{
  old_val: {
    ...
  }
  new_val: {
    ...
  }
}
```

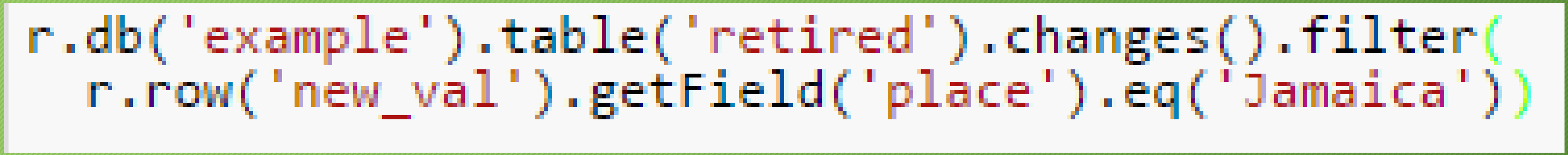
Examples



```
r.db('example').table('retired').changes()
```



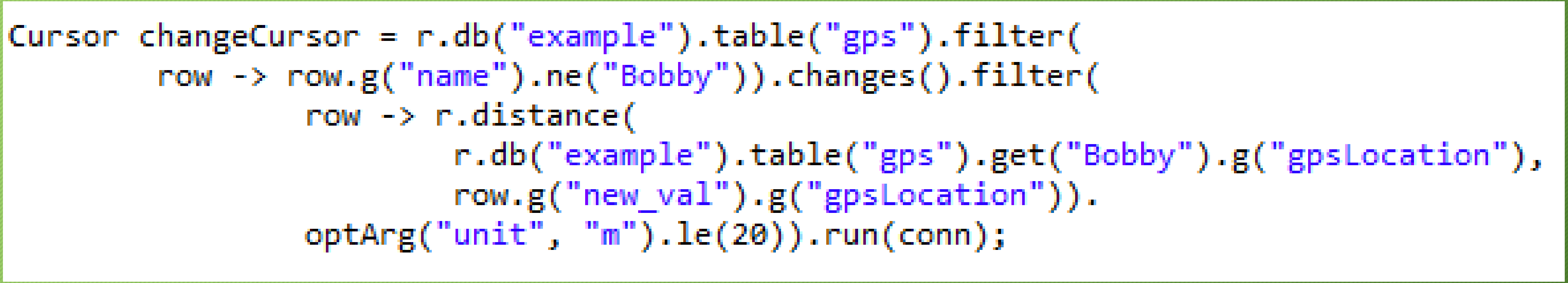
```
r.db('example').table('retired').filter(
   r.row('name').eq('Bobby')).changes()
```



```
r.db('example').table('retired').changes().filter(
   r.row('new_val').getField('place').eq('Jamaica'))
```

A Java Example

- Uses both changefeeds and GPS coordinates to monitor a table for updates in geospatial range to ‘Bobby’



```java
Cursor changeCursor = r.db("example").table("gps").filter(
        row -> row.g("name").ne("Bobby")).changes().filter(
                row -> r.distance(
                        r.db("example").table("gps").get("Bobby").g("gpsLocation"),
                        row.g("new_val").g("gpsLocation")).
                optArg("unit", "m").le(20)).run(conn);
```

Changefeeds

N

Applications

Chang Xu

When Is RethinkDB A Good Choice?

- Applications could benefit from real-time data feeds
- Use cases include:
  - Collaborative web and mobile apps
  - Streaming analytics apps
  - Multiplayer games
  - Real-time marketplaces
  - Connected devices

When Is RethinkDB NOT A Good Choice?

- ACID support or strong schema enforcement is required
  - better off using a relational database like MySQL or PostgreSQL
- Deep, computationally-intensive analytics
  - better off using a system like Hadoop or a column-oriented store like Vertica
- High write availability is critical
  - better off with a Dynamo-style system like Riak

Who Is Using RethinkDB?

- Technology startups, consulting studios
- Over 100,000 developers, hundreds of contributors

jive work better together™

NASA

Narrative

cmune

socialradar

mediafly

wise.io

Platzi

Jive Software

- A provider of business communication and collaboration solutions
- Uses RethinkDB to power its reactive business messaging app Chime

Jive Chime: Team Communications

By Jive Software

Open iTunes to buy and download apps.

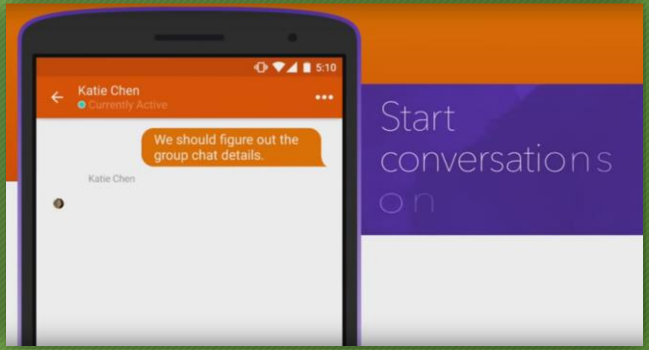
Description

Jive Chime lets you quickly and
With one-to-one and group me
responses, or be tied to your de

Jive Software Web Site ▸ Jive C

What's New in Version

– File preview improvements



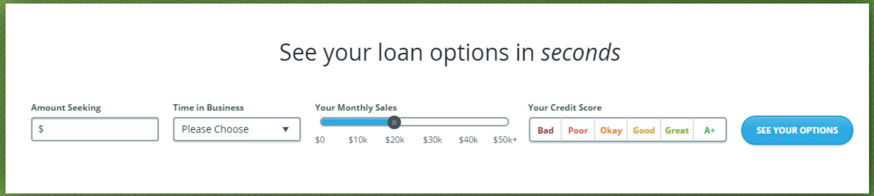
Jive Software

- Problem: requires real-time transaction and data consistency
- Solution: found RethinkDB capable of achieving these demands and easy to create powerful queries with the ReQL
- Story available on YouTube

Lendio

- Lendio helps small business owners find and secure loans for their companies
- Uses RethinkDB to process data from their clients and match loan recommendations

lendio


See your loan options in *seconds*

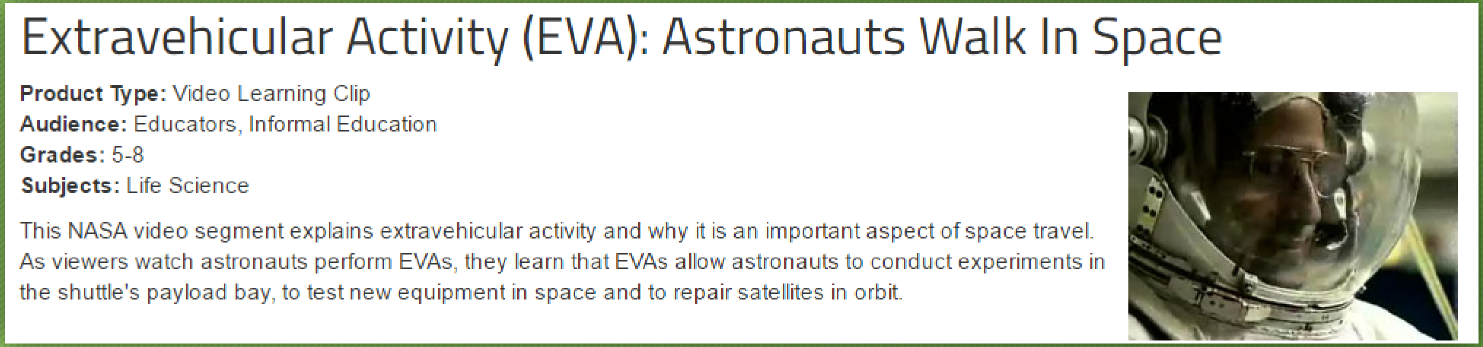
Amount Seeking: $
Time in Business: Please Choose
Your Monthly Sales: $0 $10k $20k $30k $40k $50k+
Your Credit Score: Bad Poor Okay Good Great A+

Lendio

- Problem: process real-time data from borrowers and lenders, avoiding the overhead incurred by polling
- Solution: RethinkDB’s changefeeds automatically push data to clients in real-time, and offers a compelling mix of relational features and schema less flexibility
- Story available on YouTube

NASA

- EVA (Extra-Vehicular Activity) office at NASA is working on integrating spacesuit data that stored in old database systems into a new database
- Uses RethinkDB as the new document based No-SQL database system


Extravehicular Activity (EVA): Astronauts Walk In Space

Product Type: Video Learning Clip
Audience: Educators, Informal Education
Grades: 5-8
Subjects: Life Science

This NASA video segment explains extravehicular activity and why it is an important aspect of space travel. As viewers watch astronauts perform EVAs, they learn that EVAs allow astronauts to conduct experiments in the shuttle's payload bay, to test new equipment in space and to repair satellites in orbit.

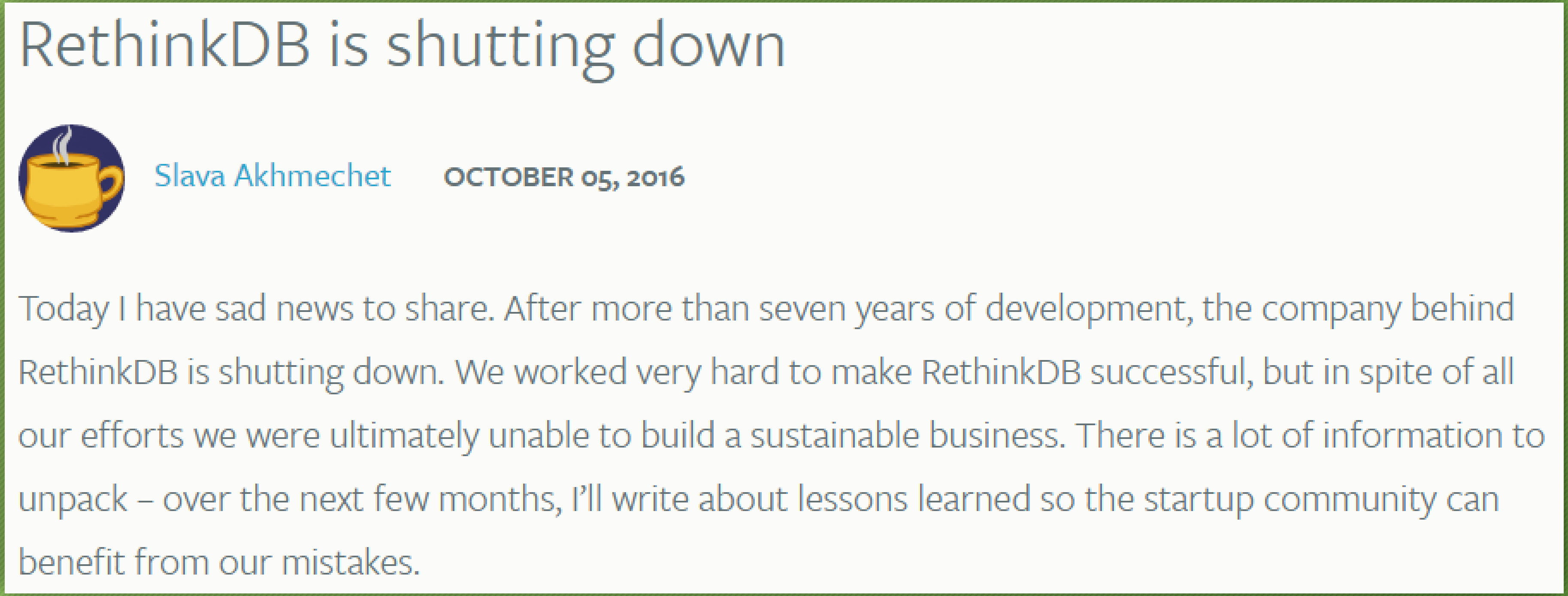
NASA

- Problem: while spacesuit data is migrating from dozens of very old databases to the new infrastructure, day to day operations/activities must not be affected
- Solution: a strategy designed around RethinkDB's changefeeds allows system to be built with data sync
- Story available on RethinkDB Website and YouTube

RethinkDB Seems to Have A Great Future

- But…

Sad News

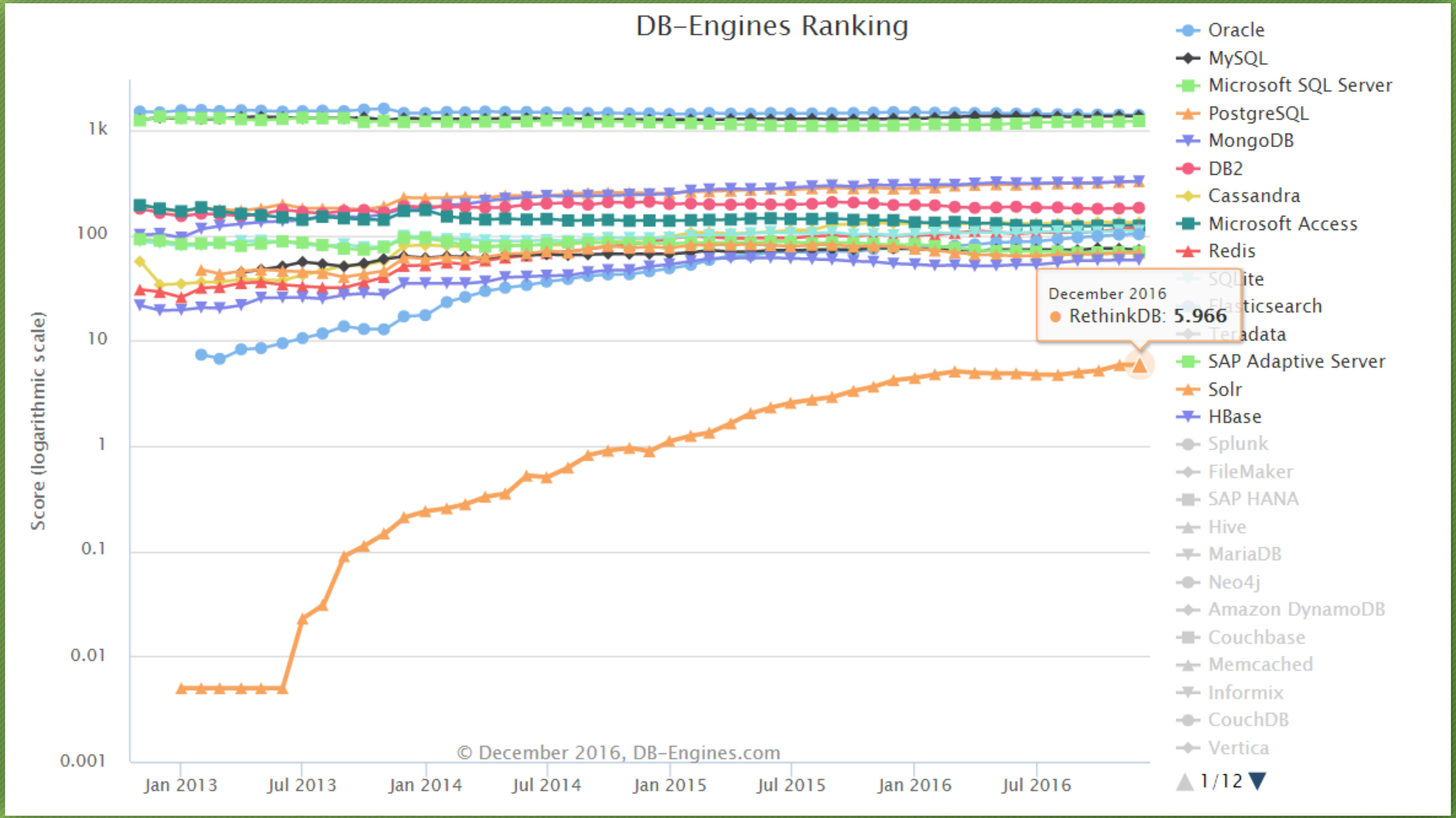


RethinkDB is shutting down

Slava Akhmechet OCTOBER 05, 2016

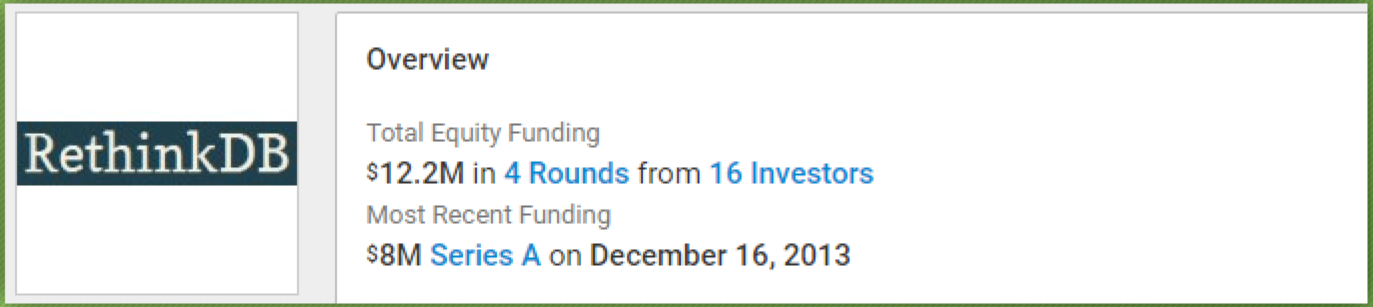
Today I have sad news to share. After more than seven years of development, the company behind RethinkDB is shutting down. We worked very hard to make RethinkDB successful, but in spite of all our efforts we were ultimately unable to build a sustainable business. There is a lot of information to unpack – over the next few months, I'll write about lessons learned so the startup community can benefit from our mistakes.

Why RethinkDB Failed



Possible Causes

- Cruel competition
- Trivial market place
- Lack of business mode
- Run out of money



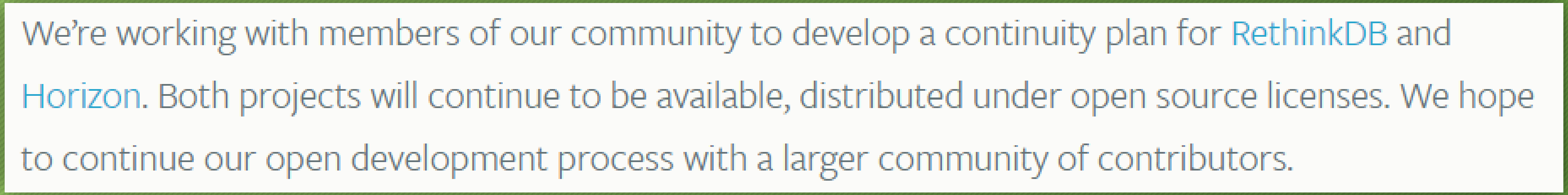
Overview

Total Equity Funding
$12.2M in 4 Rounds from 16 Investors
Most Recent Funding
$8M Series A on December 16, 2013

crunchbase

Future of RethinkDB

- RethinkDB will continue to be an open-source project


We're working with members of our community to develop a continuity plan for RethinkDB and Horizon. Both projects will continue to be available, distributed under open source licenses. We hope to continue our open development process with a larger community of contributors.

Slava Akhmechet

Cofounder of RethinkDB

Let It Go

- Let it go, let it go. Can’t think about it anymore…

Thank You