RICE®

Web Development

COMP 431 / COMP 531

Lecture 19: Databases

Instructor: Mack Joyner

Department of Computer Science, Rice University

mjoyner@rice.edu

http://www.clear.rice.edu/comp431

Part II – Back End Development

Homework Assignment 6
(Draft Back-End)
2 separate git repos
Due Thursday 11/16

In-Class Exercise 17: "Unit" Test the Back-End

- For reference the example from the slides
  https://www.clear.rice.edu/comp431/sample/RiceBookServer/src/hello.js
  https://www.clear.rice.edu/comp431/sample/RiceBookServer/src/hello.spec.js

- Download:
  https://www.clear.rice.edu/comp431/sample/RiceBookServer/src/articles.spec.js
- Implement the four tests in *articles.spec.js*
  - We're doing test-driven-development
- Implement in articles.js two endpoints (maybe you already have these)
  GET /articles/:id
  POST /article
- Your implementations should pass the tests

/inclass-17/{articles, articles.spec}.js

In-Class Exercise 17: Articles.js

articles.js

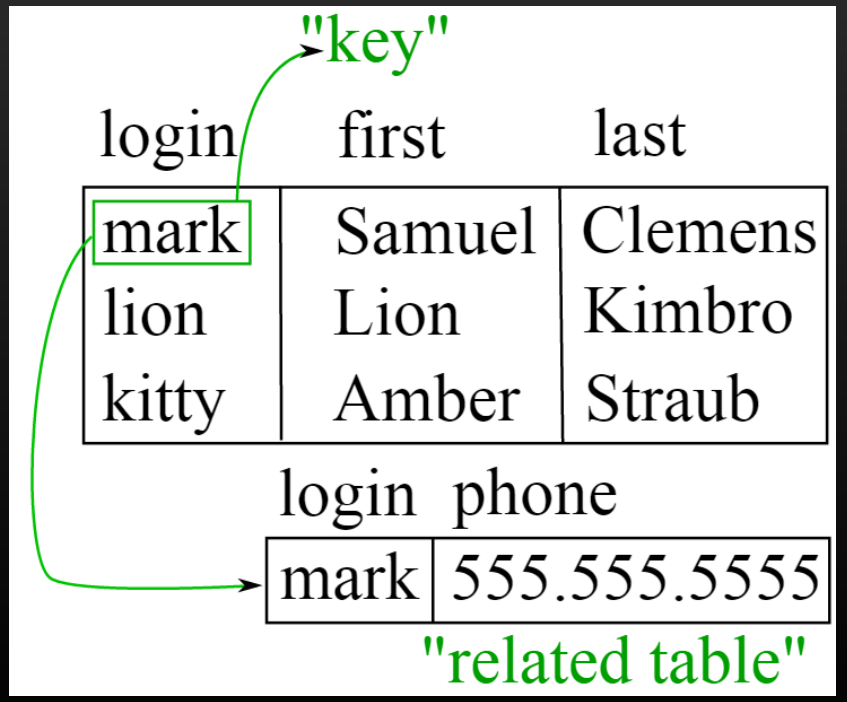
```
  },
  {
    'id': 3,
    'author': 'liam',
    'text': 'I do not know how to write'
  }]

const addArticle = (req, res) => {
    const newId = articles.length + 1;
    const newArticle = {'id': newId, 'author': req.body.author, 'text': req.body.text}
    articles.push(newArticle)
    res.send(newArticle)
}

const getArticles = (req, res) => {
    const id = req.params.id

    if (!id) {
      res.send(articles)
    }
    else {
      const article = articles.filter(a => a.id == id)
      //send first article matching id
      if (article.length > 0) {
        res.send(article[0])
      }
    }
}

const app = express()
app.use(bodyParser.json())
app.post('/article', addArticle)
app.get('/articles/:id*?', getArticles)
```

Data, data, data

- It's all about the data
- Entities
  - A "thing"
- Properties
  - The properties of the thing
- Relationships
  - Things are related to other things
- Triggers
  - When something happens do something

Databases

- A database management system (DBMS) allows for the definition, creation, querying, update, and administration of databases
- RDBMS where "R" is for relational



Relational Database

- Composed of tables
- Tables have rows (entities) and columns (fields or properties)

`prices`

column	datatype	nullable
date	datetime	NOT NULL
iid	int	NOT NULL
open	float	NOT NULL
high	float	NOT NULL
low	float	NOT NULL
close	float	NOT NULL
volume	long	NOT NULL
adj_close	float	NOT NULL

`tickers`

column	datatype	nullable
iid	int	NOT NULL
ticker	varchar(8)	NOT NULL
name	varchar(80)	NULL

Structured Query Language

```
SELECT p.date, t.ticker, p.close, p.volume
  FROM price p
  JOIN tickers t ON p.iid = t.iid
 WHERE t.ticker IN ( 'AAPL', 'GOOG' )
   AND p.date BETWEEN '2012-01-01' AND '2012-02-01'
 ORDER BY p.date DESC, t.ticker
 LIMIT 6 ;
```

date	ticker	close	volume
2012-02-01	AAPL	456.19	9644500
2012-02-01	GOOG	580.83	2320700
2012-01-31	AAPL	456.48	13988700
2012-01-31	GOOG	580.11	2142400
2012-01-30	AAPL	453.01	13547900
2012-01-30	GOOG	577.69	2330500

"SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed database engine in the world. The source code for SQLite is in the public domain"

```
> sqlite3 db.sqlite3
SQLite version 3.8.11.1 2015-07-29 20:00:57
Enter ".help" for usage hints.
sqlite> .tables
prices   tickers
sqlite> select * from prices limit 10;
2000-02-01|1|104.0|105.0|100.0|100.25|11380000|24.96
2000-02-01|4|67.5|70.63|64.37|67.44|13404600|67.44
```

SQLite from Node

```
var sqlite3 = require('sqlite3').verbose()
var memdb = new sqlite3.Database(':memory:')
var dskdb = new sqlite3.Database('db.sqlite3')

// needs to be serial because we add then query
memdb.serialize(function() {
    memdb.run("CREATE TABLE summary ("
        + "  month TEXT, iid INTEGER, ticker TEXT,"
        + "  total_dollar_volume REAL, count INTEGER"
        + ")");

    var p = "INSERT INTO summary VALUES (?, ?, ?, ?, ?)"
    dskdb.all(
        "SELECT strftime('%Y-%m-01', date) as month, "
        +"  p.iid, t.ticker, "
        +"  sum(p.close * p.volume) as total_dollar_volume, "
        +"  count(*) as count"
        +" FROM prices p JOIN tickers t ON p.iid = t.iid"
        +" GROUP BY strftime('%Y%m', date), p.iid "
        +" ORDER BY 1, 2"
        , [], function(err, rows) {
            var stmt = memdb.prepare(p);
```

SQLite from Node

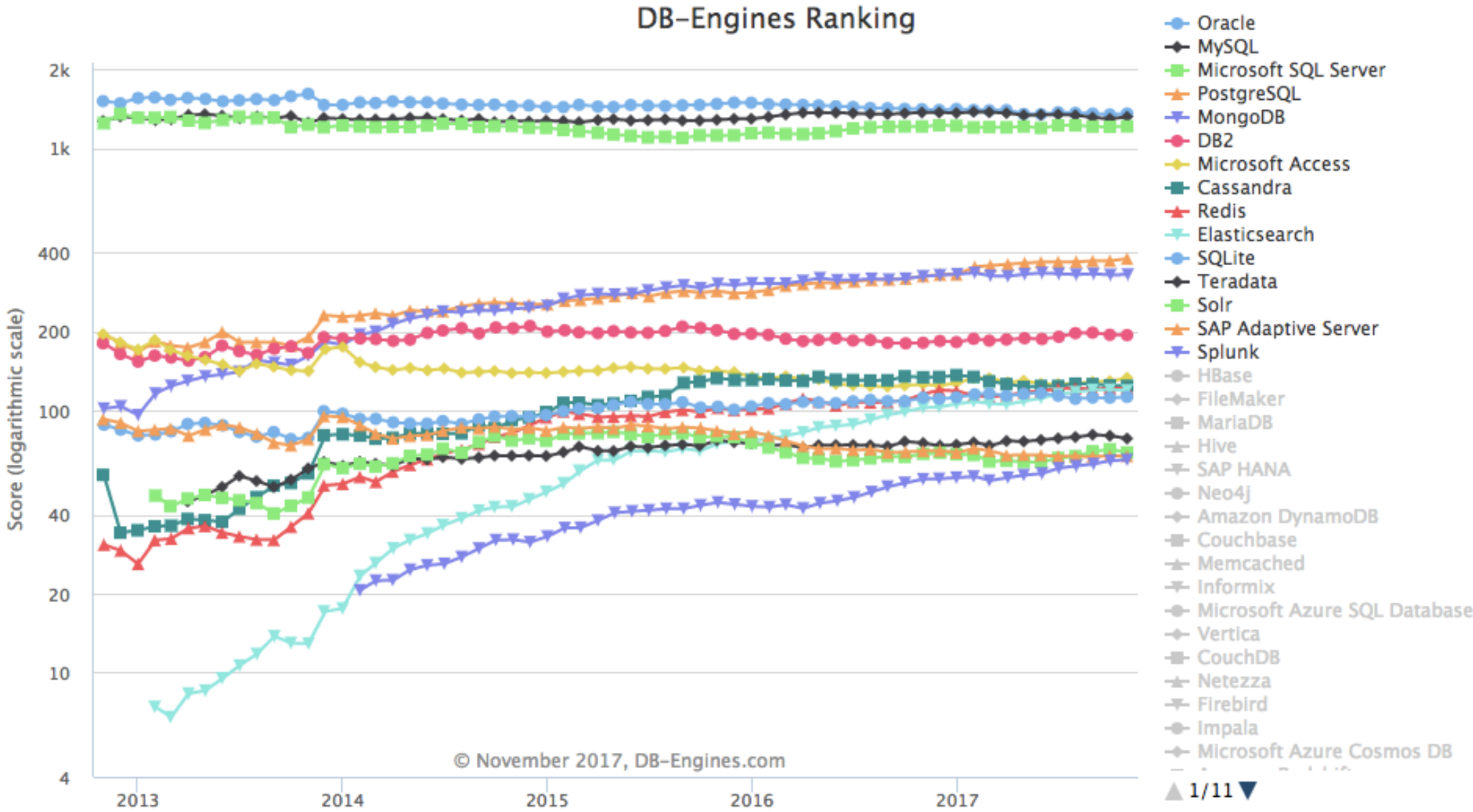
```
, [], function(err, rows) {
    var stmt = memdb.prepare(p);
    rows.forEach(function(row) {
        stmt.run([row.month, row.iid, row.ticker, row.total_dollar_volume, row.count])
    })
    stmt.finalize(function() {
        // here we are in callback hell
        console.log('month    \tticker\ttotal$vol\tavg$vol')
        memdb.each("SELECT * FROM summary WHERE ticker in ('AAPL', 'GOOG')"
            + " AND month >= '2012-01-01' LIMIT 6", function(err, row) {
          console.log(row.month + '\t' + row.ticker
            + '\t ' + parseFloat(row.total_dollar_volume/1e9).toFixed(4)
            + '\t ' + parseFloat(row.total_dollar_volume/row.count/1e9).toFixed(4));
        })
        memdb.close();
        dskdb.close();
    })
```

month	ticker	total$vol	avg$vol
2012-01-01	AAPL	105.5770	5.2788
2012-01-01	GOOG	45.0750	2.2538
2012-02-01	AAPL	204.6089	10.2304
2012-02-01	GOOG	28.7218	1.4361
2012-03-01	AAPL	322.1652	14.6439
2012-03-01	GOOG	29.6942	1.3497

Using databases asynchronously

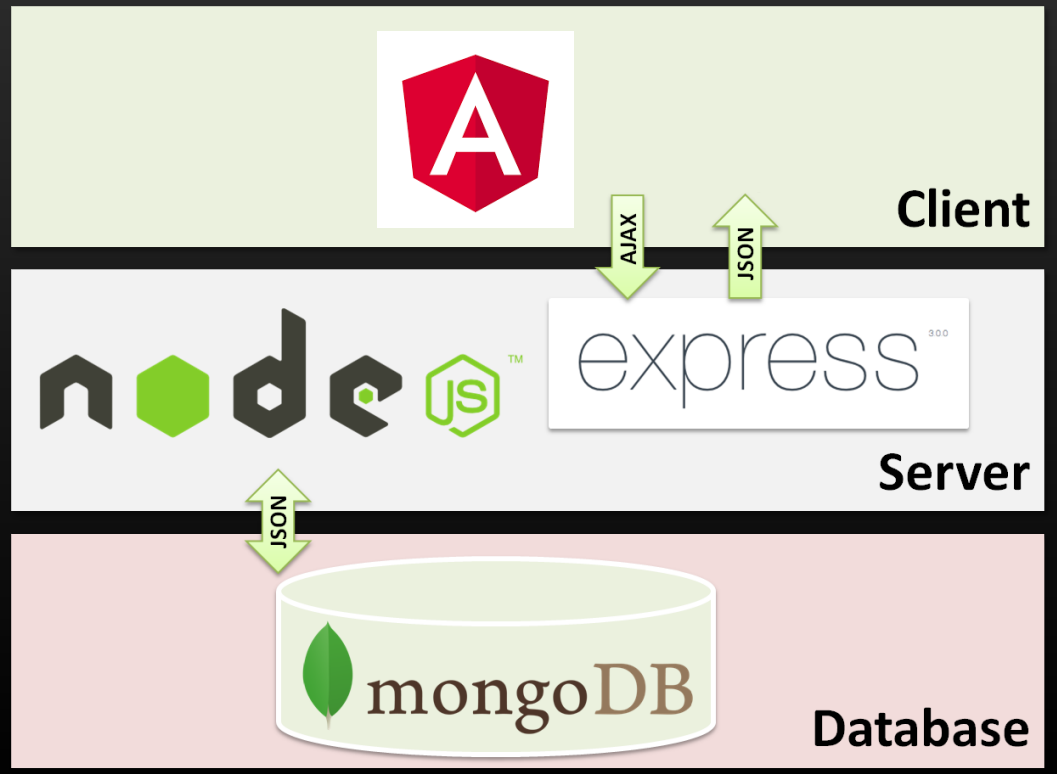
```
function countStatus(user, callback) {
    db.query("SELECT count(*) as count FROM statuses WHERE username=?", [user])
        .then(function(result) {
            callback(result[0].count)
        })
}
```

```
function countAll(req, res) {
    var payload = { users: 0, statuses: 0 }
    function queryStatuses(callback) {
        db.query("SELECT count(*) as count FROM statuses")
            .then(function(result) {
                payload.statuses = result[0].count
                callback()
            })
    }
    function queryUsers(callback) {
        db.query("SELECT count(*) as count FROM users")
            .then(function(result) {
                payload.users = result[0].count
                callback()
            })
    }
    queryStatuses(function() {
        queryUsers(function() {
            res.send(payload)
        })
    })
}
```

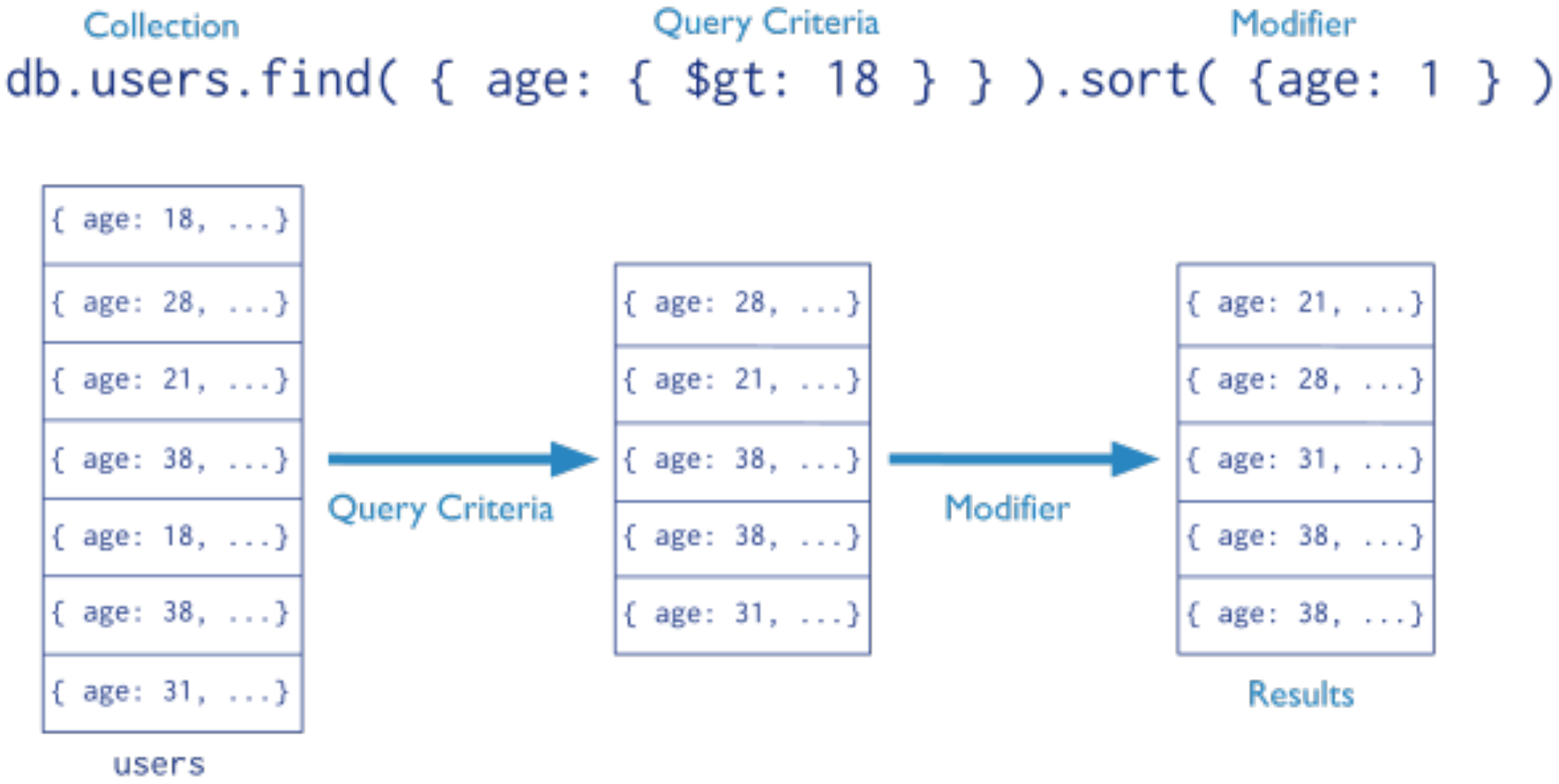

DB-Engines Ranking
Score (logarithmic scale)
2k
1k
400
200
100
40
20
10
4
2013
2014
2015
2016
2017
© November 2017, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
MongoDB
DB2
Microsoft Access
Cassandra
Redis
Elasticsearch
SQLite
Teradata
Solr
SAP Adaptive Server
Splunk
HBase
FileMaker
MariaDB
Hive
SAP HANA
Neo4j
Amazon DynamoDB
Couchbase
Memcached
Informix
Microsoft Azure SQL Database
Vertica
CouchDB
Netezza
Firebird
Impala
Microsoft Azure Cosmos DB
1/11

MongoDB

- Designed for humongous volumes of data
- Document-oriented (not row+column)
- Schema-less
  - Extensible
  - No "error" checking…
- JSON oriented
  - Binary JSON (BSON)
- Documents within Documents



MongoDB Query



MongoDB in Node

```
var MongoClient = require('mongodb').MongoClient
var url = 'mongodb://localhost:27017/webdev'
function mc(execute) {
    var args = Array.prototype.slice.call(arguments, 1)
    MongoClient.connect(url, function(err, db) {
        if (err) {
            console.err('There was a problem', err)
        } else {
            args.unshift(db)
            execute.apply(null, args)
        }
    })
}
exports.mc = mc
```

Insert some documents

```
var MongoClient = require('mongodb').MongoClient
var url = 'mongodb://localhost:27017/webdev'
function mc(execute) {
    var args = Array.prototype.slice.call(arguments, 1)
    MongoClient.connect(url, function(err, db) {
        if (err) {
            console.err('There was a problem', err)
        } else {
            args.unshift(db)
            execute.apply(null, args)
        }
    })
}
exports.mc
```

```
mc(insertDocs)
```

```
function insertDocs(db) {
    request('https://webdev-dummy.herokuapp.com/sample', function(err, res, body) {
        var posts = JSON.parse(body).posts
        // put these into a collection
        var c = db.collection('posts', function() { })
        c.insert(posts, {w:1}, function(err, result) {
            db.close()
        })
    })
}
```

Query documents

```
function queryByAuthor(db, author) {
    var c = db.collection('posts')
    c.find({ author: author }).toArray(function(err, items) {
        console.log('There are ' + items.length + ' entries for ' + author)
        var totalLength = 0
        items.forEach(function(post) {
            totalLength += post.body.length
        })
        console.log('average length', totalLength / items.length)
        db.close()
    })
}

mc(queryByAuthor, 'mrj1')
mc(queryByAuthor, 'jmg3')
```

```
There are 16 entries for mrj1
average length 428.8125
There are 16 entries for jmg3
average length 428.126
```

```
exports.mc = mc
```

MongooseJS: *an Object Document Model*

```
1  var mongoose = require('mongoose')
2  var url = 'mongodb://localhost:27017/webdev'
3  mongoose.connect(url)
4  function done() {
5      mongoose.connection.close()
6  }
7
8  var commentSchema = new mongoose.Schema({
9      commentId: Number, author: String, date: Date, body: String
10 })
11 var postSchema = new mongoose.Schema({
12     id: Number, author: String, img: String, date: Date, body: String,
13     comments: [ commentSchema ]
14 })
15 var Post = mongoose.model('post', postSchema)
16
```

```
exports.Post = Post
```

Direct *vs* ODM

```
function queryByAuthor(db, author) {
    var c = db.collection('posts')
    c.find({ author: author }).toArray(function(err, items) {
        console.log('There are ' + items.length + ' entries for ' + author)
        var totalLength = 0
        items.forEach(function(post) {
            totalLength += post.body.length
```

```
function findByAuthor(author, callback) {
    Post.find({ author: author }).exec(function(err, items) {
        console.log('There are ' + items.length + ' entries for ' + author)
        var totalLength = 0
        items.forEach(function(post) {
            totalLength += post.body.length
        })
        console.log('average length', totalLength / items.length)
        callback()
    })
}
```

Other Solutions

SQL, noSQL, newSQL

redis

Google™ Spanner

ELASTIC SEARCH

HBASE

Bigtable

cassandra

Amazon RDS

amazon web services

Amazon DynamoDB