AMIS

INTRODUCING NOSQL AND MONGODB

Pom Bleeksma & Lucas Jellema

14 March 2017, Nieuwegein

mongoDB

AGENDA

INTRODUCTION

WHAT'S NOT SQL ABOUT NOSQL – HISTORY, BACKGROUND, USE CASES

OVERVIEW OF NOSQL DATABASES

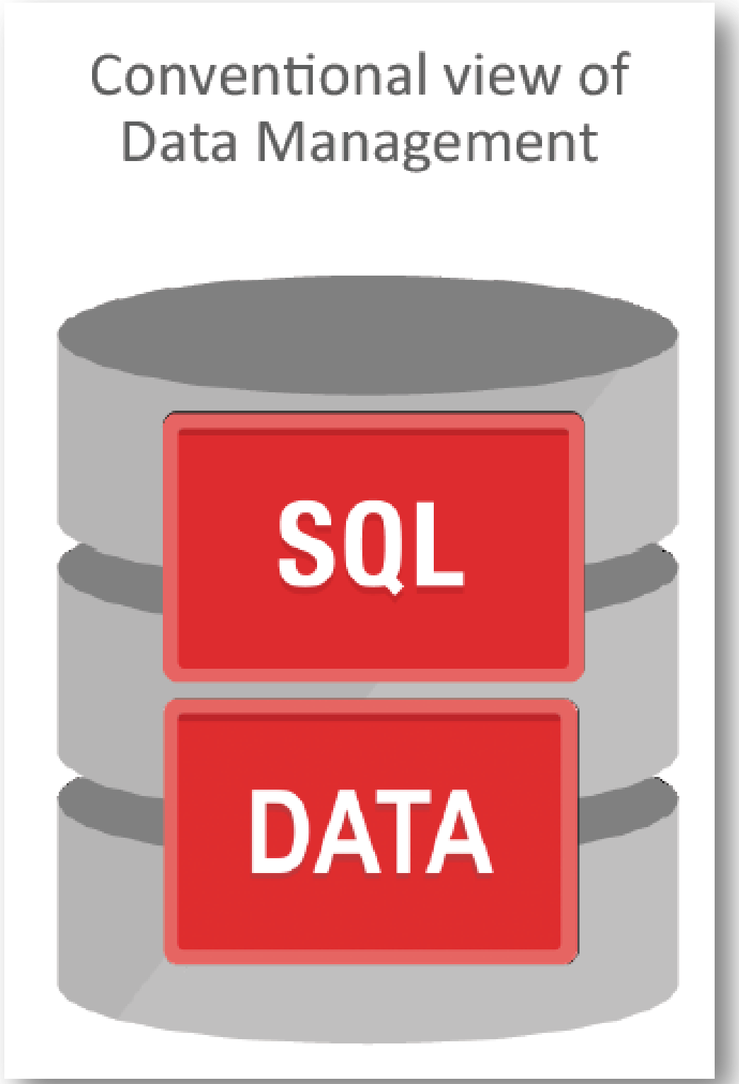
INTRODUCING MONGODB

GETTING STARTED WITH MONGODB – INSTALLATION, CONFIGURATION, ADMINISTRATION

DEVELOPING WITH MONGODB – JAVA & NODE.JS

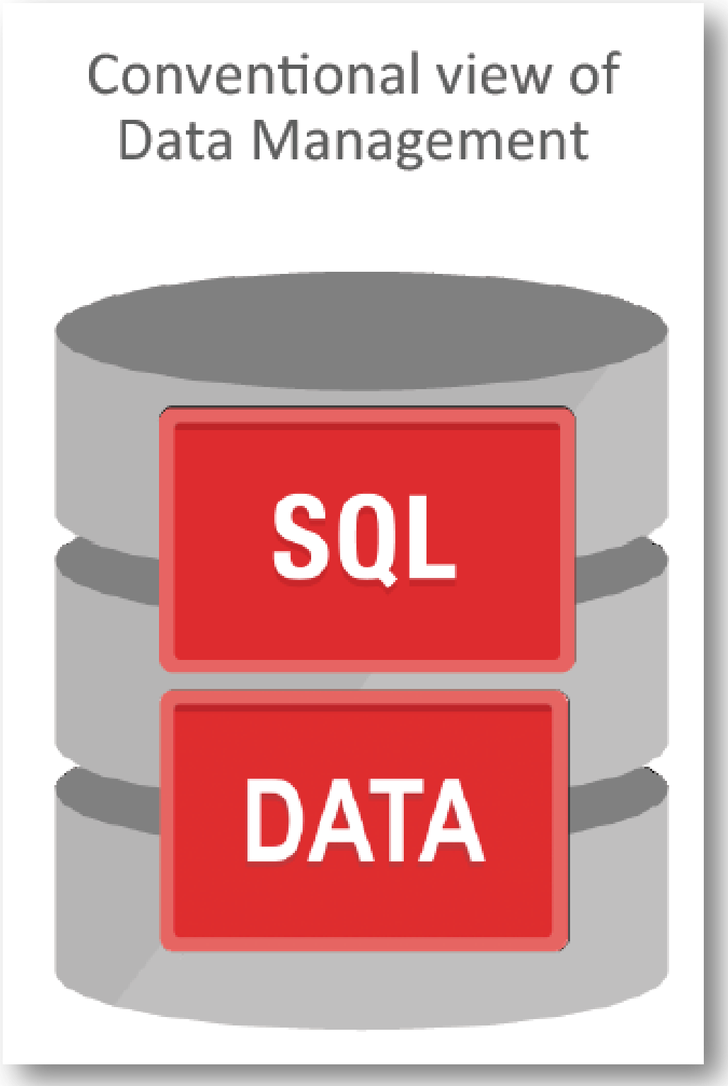
HANDSON WORKSHOP

TRADITIONAL APPROACH:
ALL DATA IN ENTERPRISE RELATIONAL DATABASE



TRADITIONAL APPROACH: WITHOUT MUCH REGARD FOR

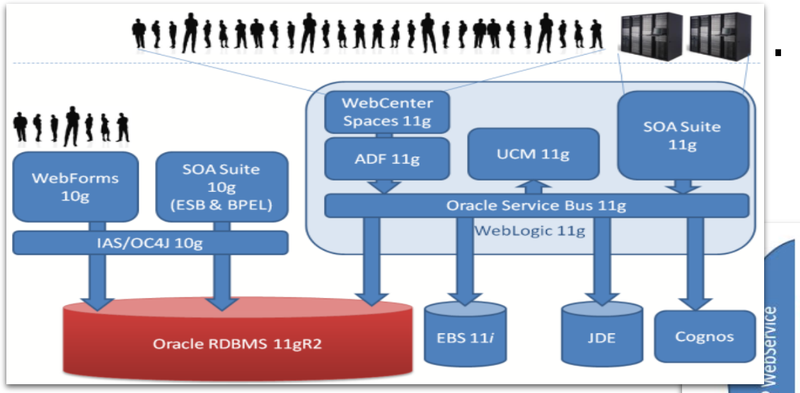
- What is it used for?
  - By whom, where, in what way, using which tools
- Ratio between writes and reads
- Ad hoc access?
- What format is it in/should it be in?
- What confidentiality & integrity is required?
- How much of it?
- How long relevant (hot vs cold vs dead)?
- How fine grained and how accurate?
- How much change?

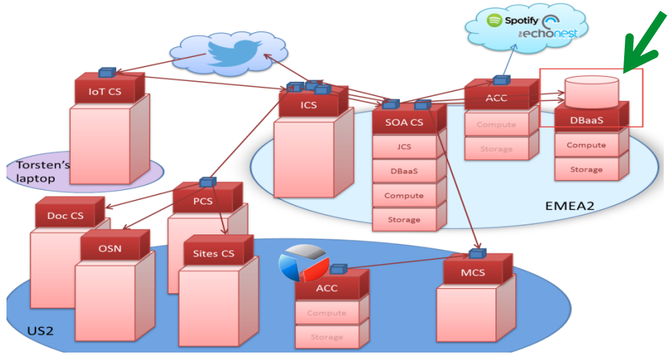


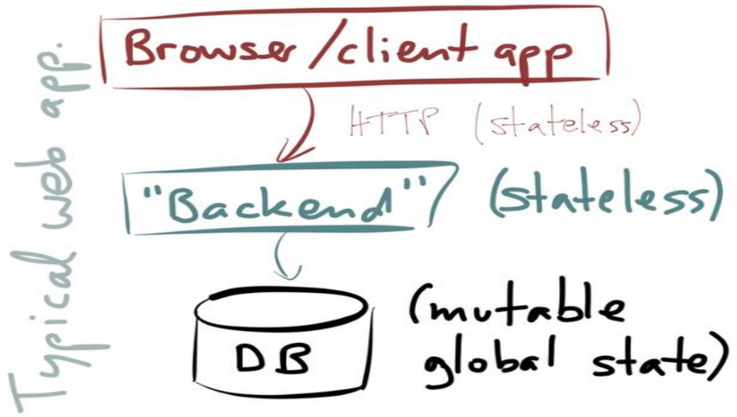
AMIS

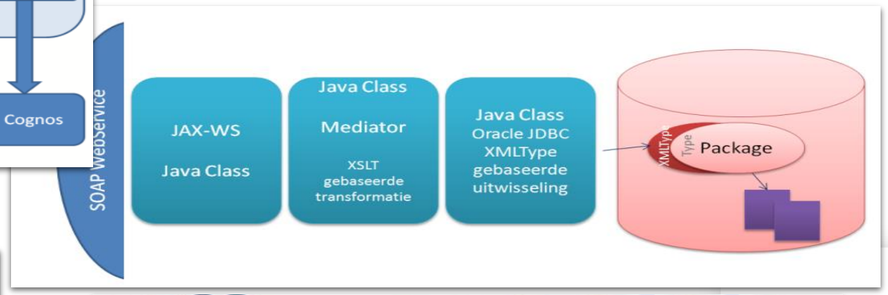
DATABASE: SINGLE POINT OF …
SCALABILITY | AVAILABILITY | CONSISTENCY

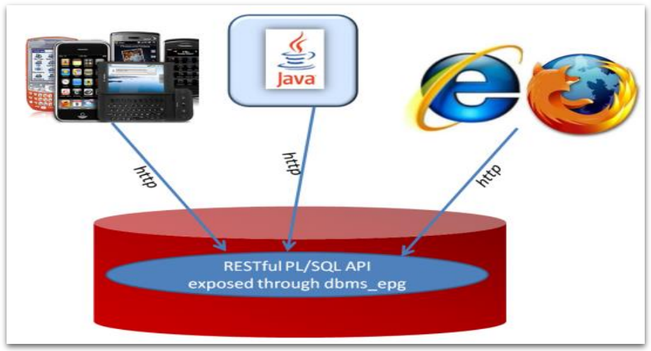
- Single Point of …

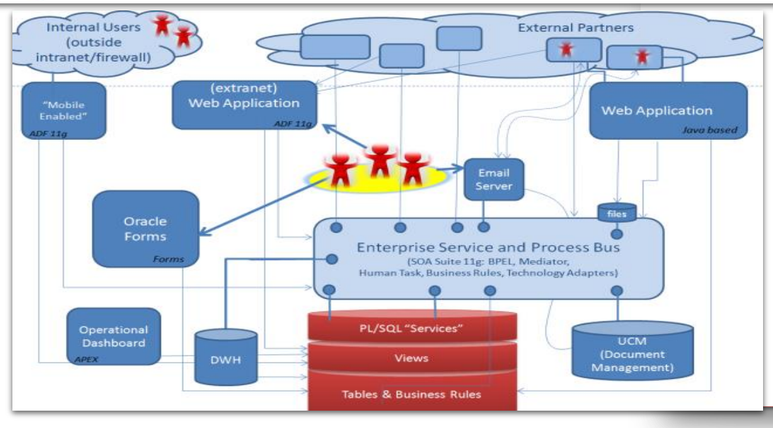


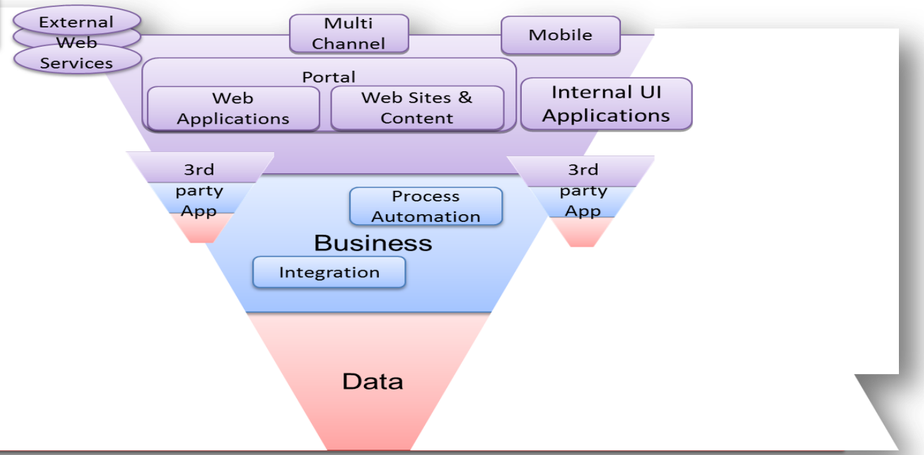






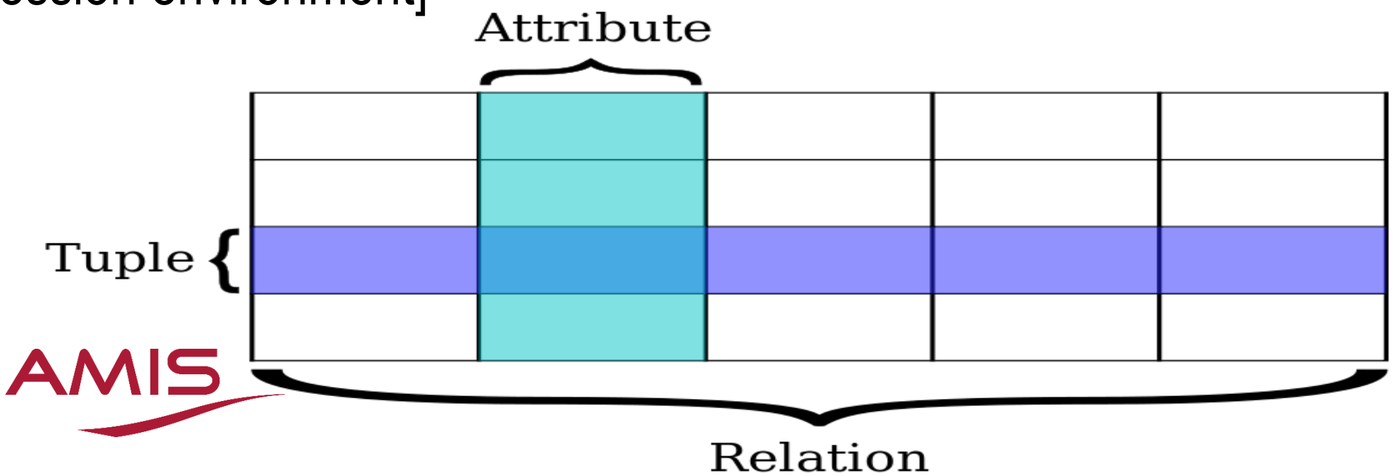






RELATIONAL DATABASES

- Based on **relational model** of data (E.F. Codd), a mathematical foundation
- Uses **SQL** for query, DML and DDL
- Transactions are **ACID** (Atomicity, Consistency, Isolation, Durability)
  - **A**ll or nothing
  - **C**onstraint Compliant
  - **I**ndividual experience [in a multi-session environment] (aka concurrency)
  - **D**own does not hurt



THE HOLY GRAIL OF NORMALIZATION

- Normalize to prevent data redundancy, discrepancies (split brain) and storage waste
- Pragmatic approaches - recognizing the fact that some data is read far more frequently than that it is created and modified
  - Materialized View
  - Virtual Column
  - Index
  - Sharding
  - Replication
  - Data Guard
  - Data Warehouse & Data Marts
  - Client Side Caching
  - In Memory Database

AMIS

ACID COMES AT A COST

- Transaction results have to be persisted in order to guarantee **D**
- Concurrency requires some degree of locking (and multi-versioning) in order to have **I**
- Constraint compliance (unique key, foreign key) means all data hangs together (as do all transactions) in order to have **C**

AMIS

ACID COMES AT A COST

- ACID means:
  - All data closely together (and sometimes far from applications and users)
  - All transactions centrally organized
  - All data persisted as part of the transaction
- ACID means:
  - Hard to scale out (horizontally)
- Oracle offers scale up with Real Application Clusters

AMIS

THE RELATIONAL MODEL IN PRACTICE

- Traditional Relational Data Model has severe impact on physical disk performance
  - Transaction Log => Sequential Write (append to file)
  - Data Blocks require much more expensive Random Access disk writes
- Indexes (B-Tree, Bitmap, …) are used to speed up query (read) performance
  - and slow down transactions
- Relational data does not [always] map naturally to data format required in application (OO, JSON, XML)
- Capability to join and construct ad-hoc queries across the entire data model is powerful
- Declarative integrity constraints allow for strict enforcement of data quality rules
  - *“the data may be non sensical, but at least it adheres to the rules, whatever the origin”*
- SQL has been embraced by huge numbers of tools & technologies and IT/Data professionals

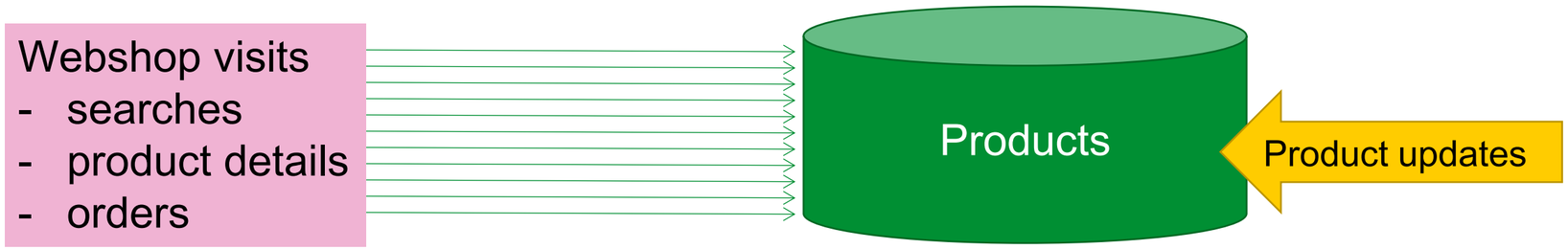
AMIS

DATABASES RE-EVALUATED

- Not all use cases require ACID (or can afford it)
  - Read only (product catalog for web shops)
  - Inserts only and no (inter-record) constraints
  - Big Data collected and “dumped” in Data Lake (Hadoop) for subsequent processing
  - High performance demands
- Not all data needs structured formats or structured querying and JOINs
  - Entire documents are stored and retrieved based on a single key
- Sometimes – scalable availability is more important than Consistency – and ACID is sacrificed
  - CAP-theorem states: Consistency [across nodes], Availability and Partition tolerance can not all three be satisfied

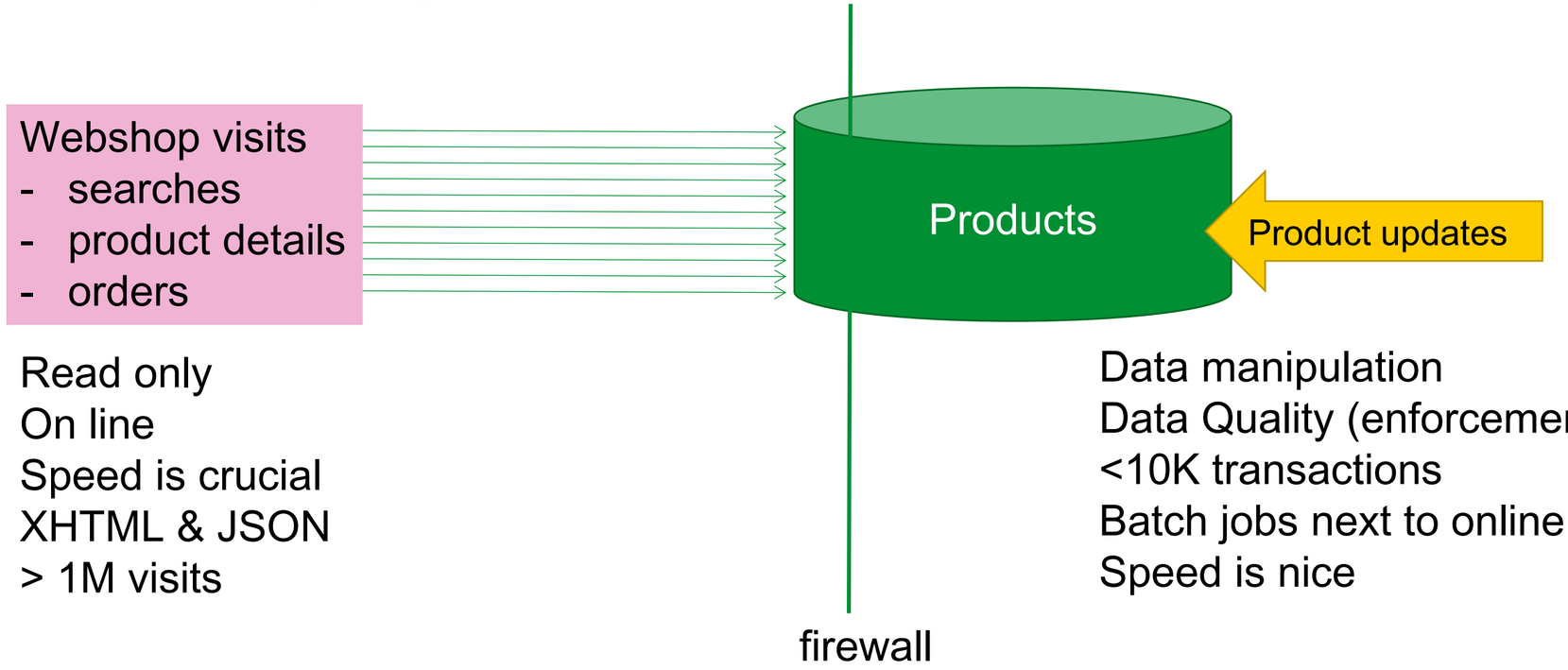
USE CASE: WEBSHOP

- Webshop – 1M visitors per day
- Product catalog consists of millions of records
  - The web shop presents: product description, images, reviews, pricing details, related offerings, stock status
- Products are added and updated and removed every day
  - Although most products do not change very frequently
  - Some vendors do bulk manipulation of product details

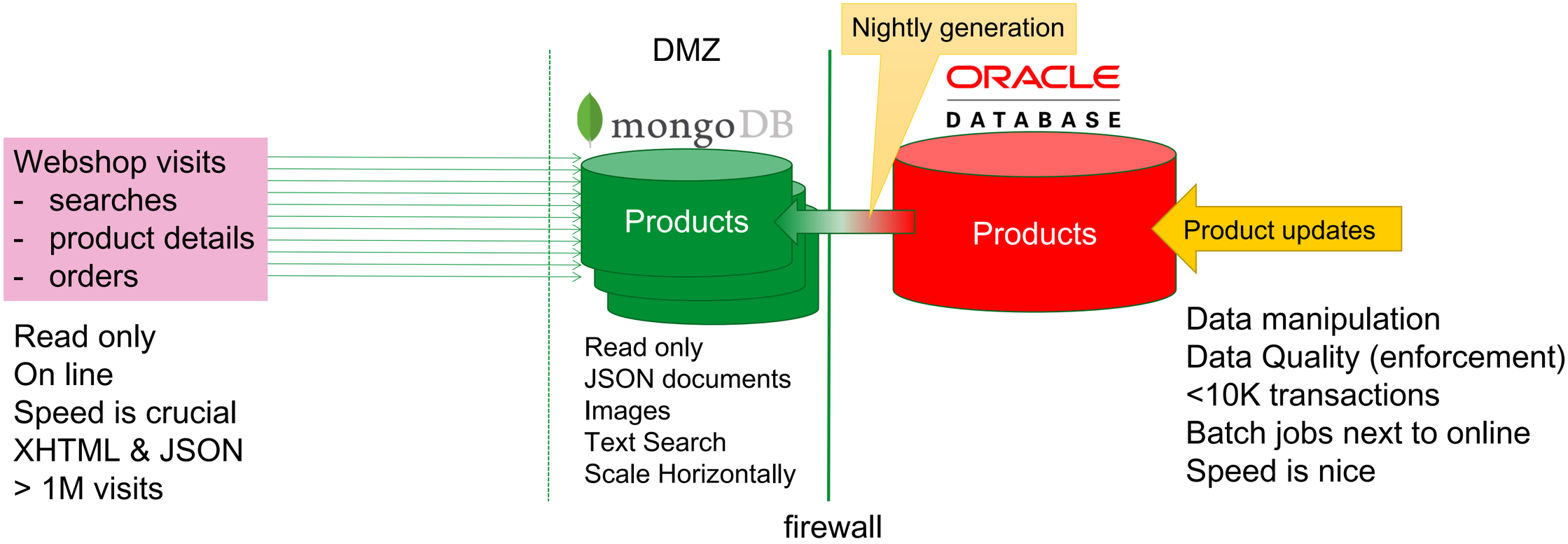


AMIS

USE CASE: WEBSHOP TECHNOLOGY AND ARCHITECTURE CONSIDERATIONS



USE CASE: WEBSHOP TECHNOLOGY AND ARCHITECTURE CONSIDERATIONS



A FEW OBSERVATIONS

- Redundancy
  - for performance, scalability, availability
- Commodity hardware for scale out architecture
- Data (documents) prepared for specific application usage (super MV)
  - Format
  - Search paths
- Time lag in data modifications is accepted
  - Perhaps even desirable to offer a consistent view to webshop users
- Transactions in one engine
  - ACID still in place
- Transactions do not compete with reads
  - vying for the same physical resources

NOSQL AND BASE

- NoSQL arose because of performance and scalability challenges with Web Scale operations
- NoSQL is a label for a wide variety of databases that aspect of a true relational database
  - ACID-ness, SQL, relational model, constraints
- The label has been used since 2009
  - Perhaps NoREL would be more appropriate
- Some well known NoSQL products are
  - Cassandra, MongoDB, Redis, CouchDB, …
- BASE as alternative to ACID:
  - **b**asically **a**vailable, **s**oft state, **e***ventually consistent (after a short duration)*

AMIS

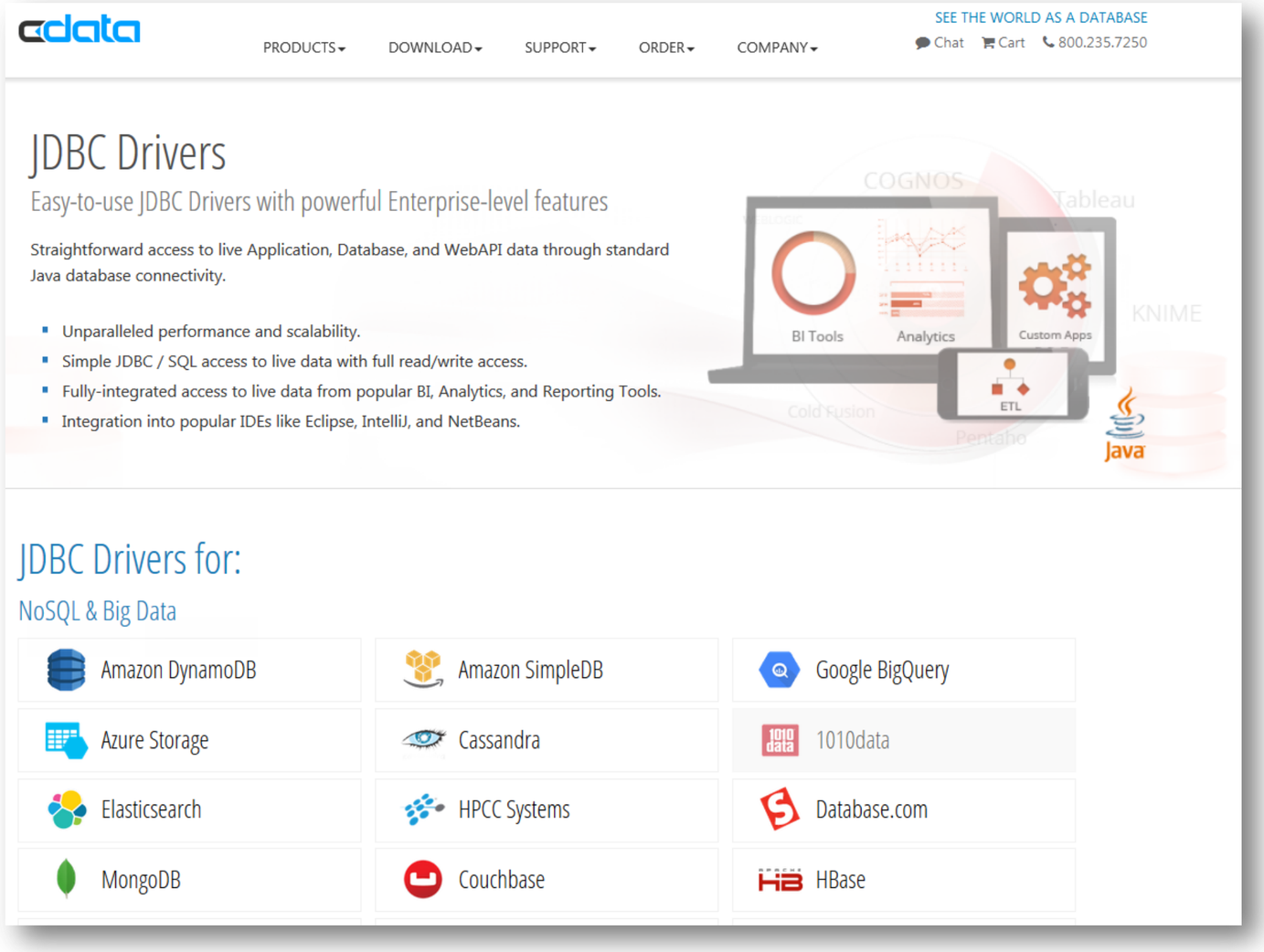
TYPICAL FOR NOSQL

- Focus on speed, availability and scalability
  - Horizontal scale out – distributed with load balancing and fail-over
- No (predefined) Data Structure
- Integrity primarily protected by application logic
- Open Source (most offerings are, not all: MarkLogic)
- Close(r) attention for how the data is used
  - Application oriented data format and search paths and specialized database per application (microservice, capability)
  - Similar to the switch from SOA to API/Microservice
- Reads (far) more relevant than writes
- Data redundancy & denormalization
- No data access through SQL

AMIS

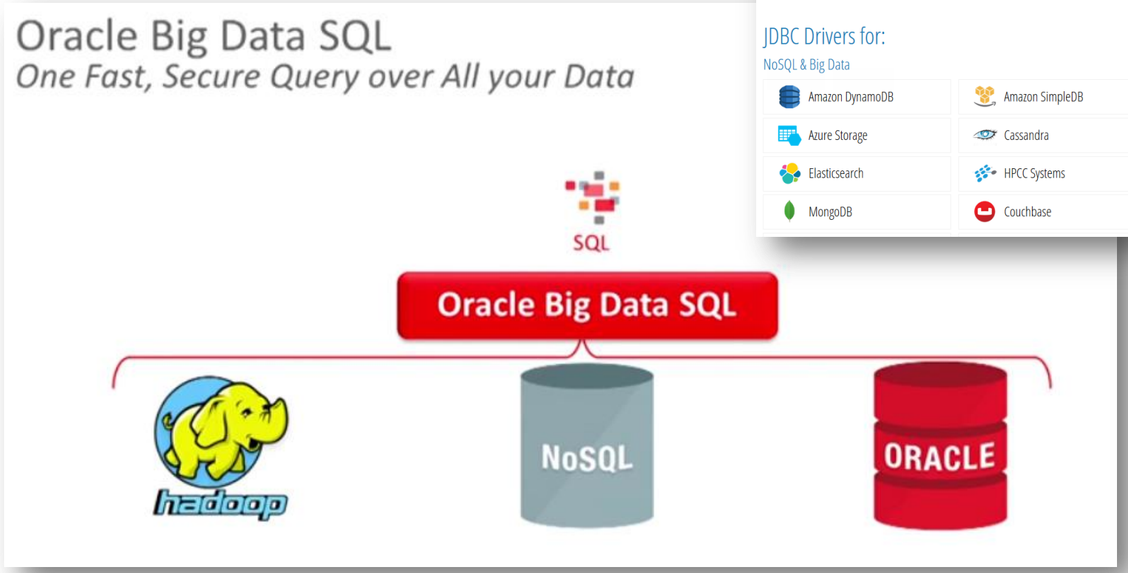
NOSQL MEANS:
NO DATA ACCESS THROUGH SQL

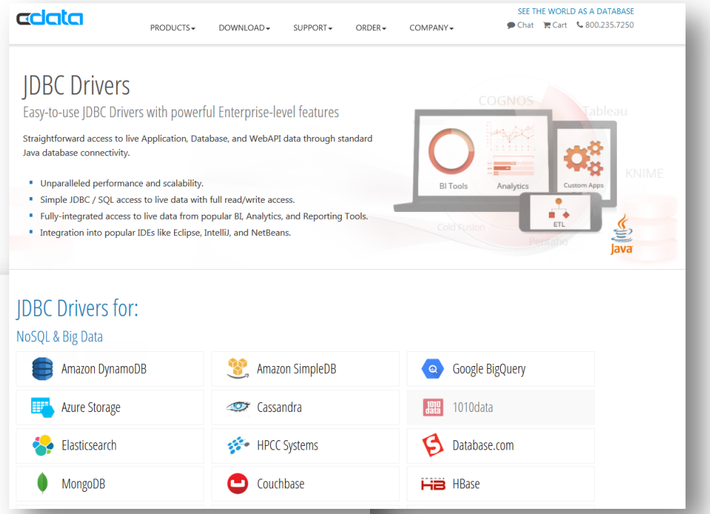
- However
  - Data Professionals and Developers speak SQL
  - Reporting, Dashboarding, ETL, BI tools speak SQL
- There is no common query language across NoSQL products



NO DATA ACCESS THROUGH SQL

- However
  - Data Professionals and Developers speak SQL
  - Reporting, Dashboarding, ETL, BI tools speak SQL
- There is no common query language across NoSQL products





- Attempts from many vendors to create drivers that translate SQL statements into NoSQL commands for the specific target database
  - To protect existing investments in SQL – skills, tools, applications, reports, ..

AMIS

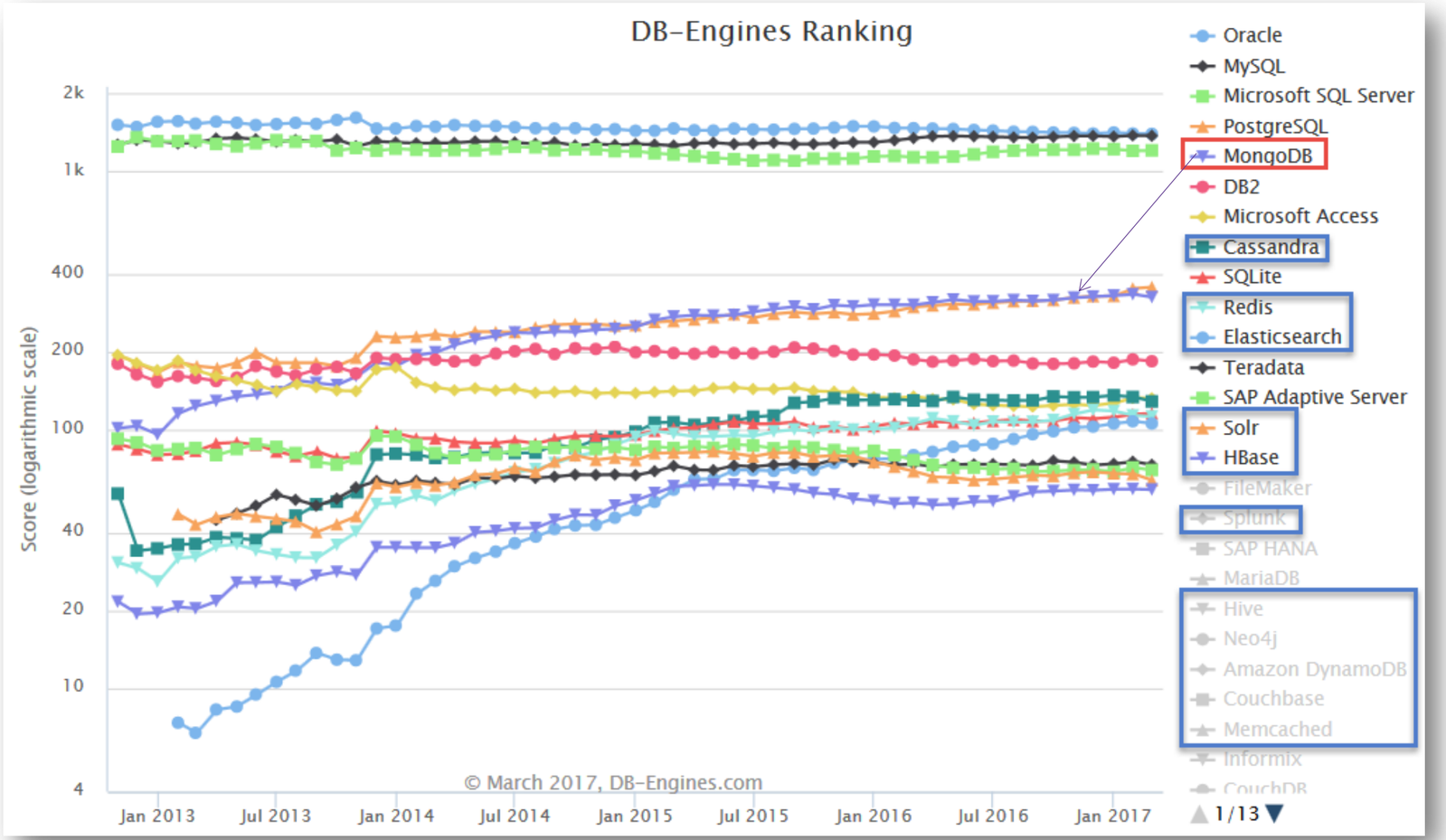
TYPES OF NOSQL DATABASES

- Key-Value (from in-memory cache to hardware appliance – Memcached, Hazelcast, Coherence, Redis, Oracle NoSQL/Berkeley DB)
- Wide Column Store (Cassandra, Google BigTable, HBase)
- Document (JSON, XML, YAML and binary forms – MongoDB, CouchDB, Couchbase, MarkLogic)
- Graph (networks of data, triplets, RDF stores – Neo4J)
- Miscellaneous: object, tabular, tuple, elastic search index, Hadoop (hdfs) style, Kafka Topic

AMIS

(LEADING) NOSQL DATABASE PRODUCTS

- MongoDB is (one of) the most popular (by any measure)
- Cloud (only):
  - Google BigTable,
  - AWS Dynamo
- Cache (in memory)
  - ZooKeeper, Redis, Memcached, …
- Hadoop/HDFS
- Oracle NoSQL (fka Berkeley DB)



AMIS

HISTORY OF MONGODB

- 10gen – startup from 2007 in New York City
  - Developed database as component in Platform as a Service product
- Initial release of MongoDB: 2009
  - 10gen changed its name to MongoDB Inc. in 2013
  - Offers enterprise support, MongoDB Atlas cloud service, MongoDB Compass, Connectors
- Current stable release: 3.4.2 (3.5.2 is in preview)
- Open Source - GNU Affero General Public License and the Apache License.
- MongoDB == Hu*mongo*us *D*ata*b*ase
- Evolution of MongoDB is still pretty fast
  - Some crucial enterprise database capabilities were added fairly recently or are still lacking
- MongoDB is number one NoSQL database in terms of popularity

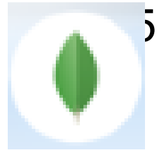
AMIS

CORE ARCHITECTURE

- MongoDB stores JSON documents in a binary format (BSON)
  - Documents are stored in collections (similar to row or records in tables)
- Interaction is through JavaScript
  - taking the place of SQL for DML and query
- Written in C++
- Has the V8 JavaScript engine included
- Runs on Mac OS X, Windows, Solaris, and most flavors of Linux
- Source code is on GitHub
- Has GeoSpatial and Tekst indexes and search capabilities
- MongoDB, Inc. officially supports drivers for C, C++, C#, Erlang, Java, Node.js, JavaScript, Perl, PHP, Python, Scala, and Ruby

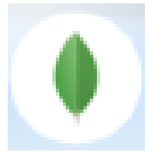
AMIS

ARCHITECTUUR, INSTALLATIE, CONFIGURATIE, OPSLAG, SHELL



- Mongo draait op windows en linux.
- Windows install met msi file
- linux install met tgz file.
- Users en directories vrij te kiezen
- Mongod --dbpath db1-- port 27001
- Default path=\data\db, default port=27017
- Mongo [-- port hostname:27001] [--shell some_json_file]
- Dbs, collections

IMPORT EXPORT



- Mongoimport
- Json
- Csv, tsv, maar hebben een descriptor file nodig

- mongoexport -h -d -c -o -q
- Leesbare output, minder geschikt voor backups

AMIS

CRUD

- show dbs
- use db
- db.<collection>.find()
- db.<collection>.find().pretty()
- select: db.find({key:value,key:value})
- present: db.<collection>.find({key:value},{key:1,key:1,_id:0})
- explain: db.find({key:value,key:value}).explain("executionStats")
- db.companies.insert({Company:"AMIS",Event:"MongoDB Workshop"})
- db.companies.update({Company:"AMIS"},{$set: {Location:"Nieuwegein"}})
- Geen commit! Ofwel, autocommit.

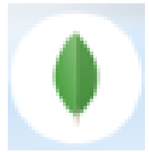
AMIS

CRUD

- db.companies.remove({"Company" : "AMIS"})
- $unset -> Om een veld uit een document te verwijderen.
- $inc -> om een numeriek veld te verhogen met een bepaalde waarde. Als het veld niet bestaat wordt het toegevoegd (met de waarde van de verhoging)
- $push -> Om een waarde aan een array toe te voegen.

AMIS

INDEXING, EXPLAIN



- _Id altijd geïndexeerd en unique
- Indexen helpen bij zoeken
- Indexen mogen compound zijn, dwz op meerdere keys gecombineerd, bv index op woonplaats+leeftijd
- db.user.find().sort({ woonplaats:1, leeftijd: -1 }) De -1 betekent aflopend geïndexeerd.
- Resultaten rechtstreeks uit index indien mogelijk (fat indexes in SQL)
- db.user.find({woonplaats:”Zoetermeer”}).explain(“executionStats”) geeft een goed overzicht van het gevolgde plan

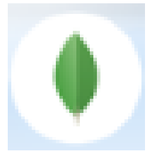
AMIS

REPLICA SETS

- Replica sets zijn gedupliceerde collections
- Beveilliging tegen uitval
- De members besluiten onderling welk member primary is. Alleen daarop kan DML plaatsvinden
- Altijd oneven aantal members, eventueel mbv een arbiter
- Secondary members kunnen wel voor select gebruikt worden, als je eerst slaveOk() ingeeft
- Mogen op dezelfde host draaien, mogen ook op verschillende hosts draaien.
- writeConcern: Je kan expliciet aangeven op hoeveel members je DML uitgevoerd moet zijn voor je je prompt terug krijgt.

AMIS

SHARDING



- Shard = splinter
- Range based partitioning
- Altijd index nodig op de shard key. Hoeft niet unique te zijn.
- Iedere shard is een proces. Bij voorkeur ieder proces zijn eigen host.
- Scale out proces
- Metadata staat in aparte config db. Typically 3 stuks.
- Clients verbinden met een mongos server. Dat is een instance zonder database die uit de config haalt waar de data zich bevindt.
- Iedere shard kan bestaan uit replica sets.

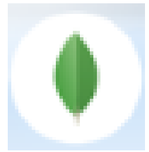
AMIS

SHARDING

- Data verdeelt zichzelf over chunks.
- Chunks kunnen splitten wanneer ze te groot worden (typically 100MB maar is aan te passen naar wens)
- Chunks kunnen migraten naar een andere chard. Mongo streeft ernaar elke shard evenveel chunks te laten hebben.
- Tijdens migration kan DML gewoon doorgaan.

AMIS

BACKUP & RESTORE

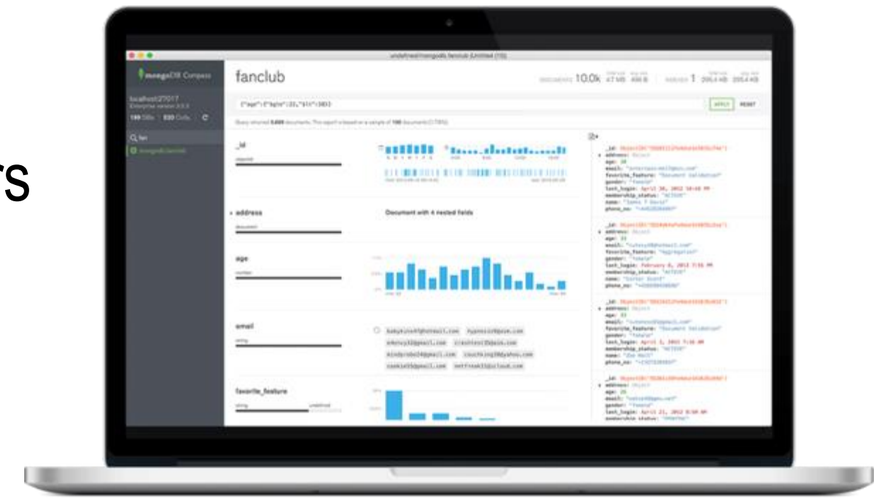


- Backup met mongodump
- Creëert een dump directory met daaronder een directory per database en plaatst daarin 1 of meerdere .bson files, een binair formaat.
- Kan vanaf ieder member van een replica set. Bij voorkeur eentje die het niet druk heeft.
- Mongo kent geen transactions, dus backup is niet consistent. Het is maar afwachten wat erin staat als er veel DML plaatsvindt.
- De tegenhanger heet mongorestore.
- Restore van bestaande database lukt niet: duplicate key op _id

AMIS

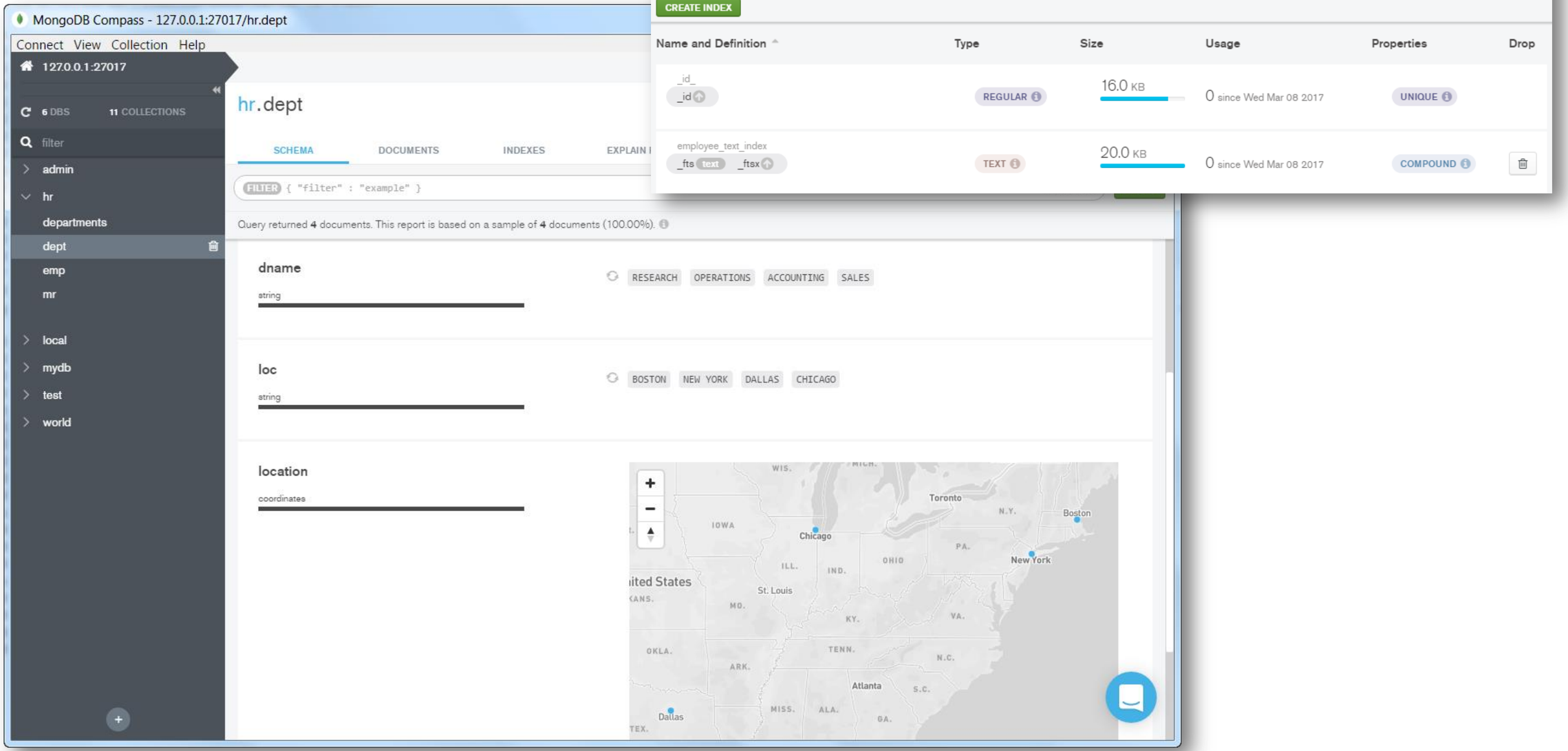
MONGODB COMPASS (AKA NOSQL DEVELOPER)

- The GUI for MongoDB
  - Visually explore data
  - Run ad hoc queries in seconds.
  - Interact with your data with full CRUD functionality
  - View and optimize your query performance.
- Separately installed product
  - On Windows, OS X, Linux, …
- Can run against local and remote MongoDB servers



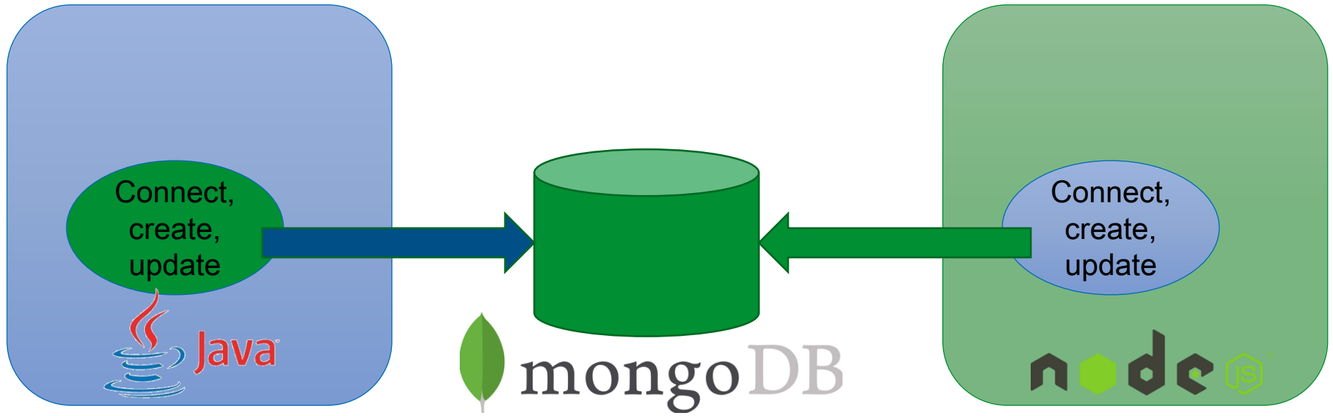
AMIS

MONGODB COMPASS



PROGRAMMING AGAINST MONGODB

- MongoDB officially supports drivers for C, C++, C#, Erlang, *Java*, *Node.js*, JavaScript, Perl, PHP, Python, Scala, and Ruby

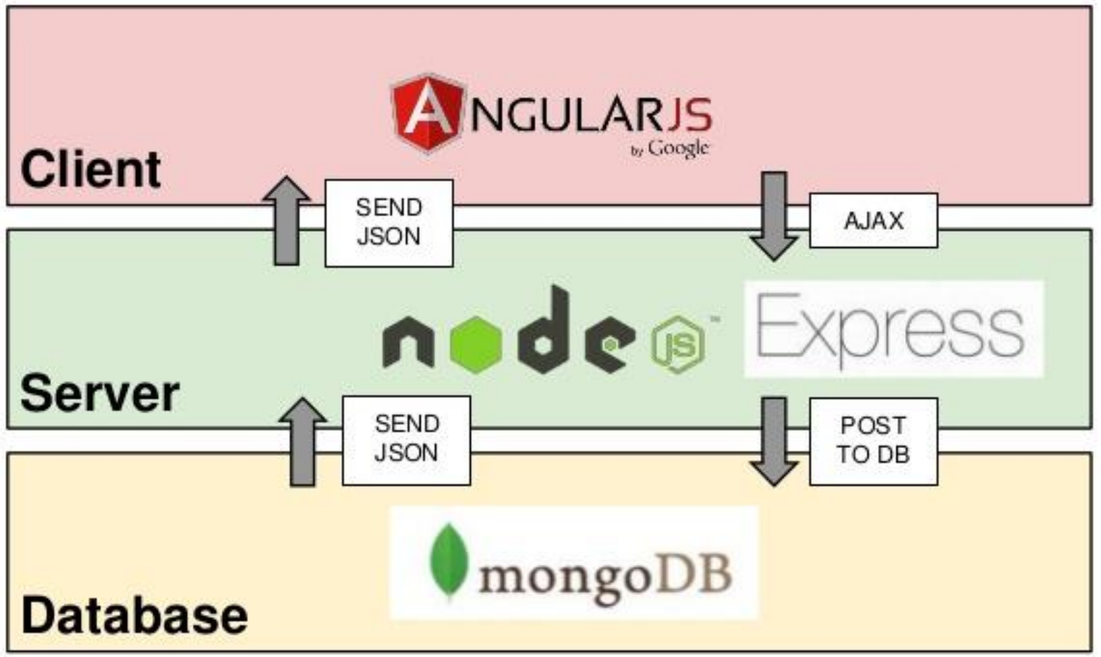


  - Note: there is also a Connector for Hadoop
- Several REST APIs for MongoDB are available from the community
  - RESTHeart (Java), Eve (Python),, Crest (Node), AMID, Kule, DreamFactory

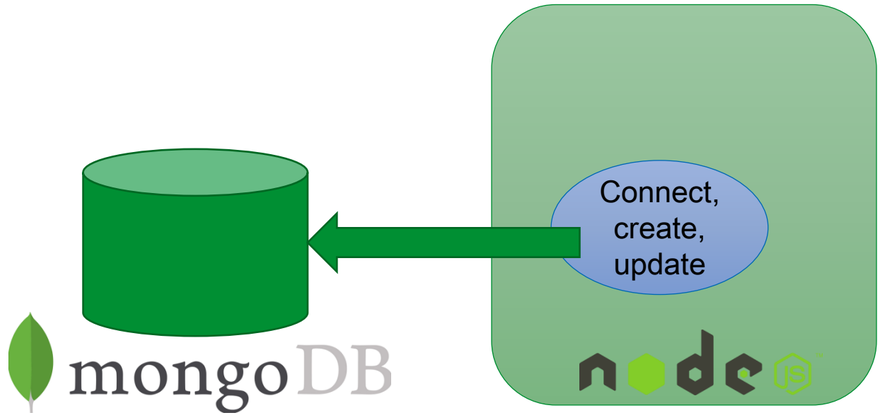
AMIS

MEAN STACK

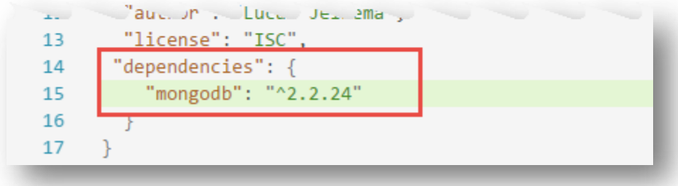
- Modelled after LAMP
- End-to-End JavaScript/JSON
- Term coined in 2013 (by MongoDB)



INTERACTING WITH MONGODB FROM NODE.JS



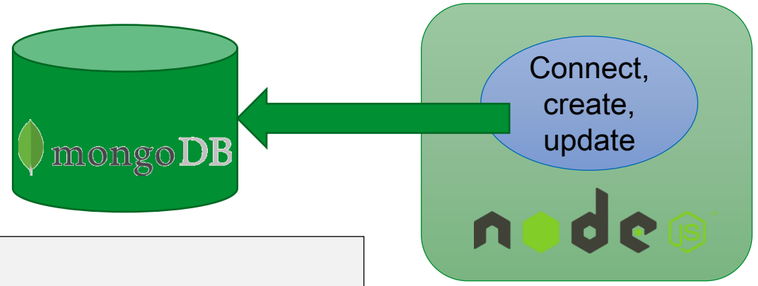
- MongoDB Node.js Driver 2.0
  - Support for ECMAScript 6.0 – Promises for asynch
  - http://mongodb.github.io/node-mongodb-native/2.0/
- Using mongodb in a NodeJS application
  - npm install mongodb –save



  - var MongoClient = require('mongodb').MongoClient;

AMIS

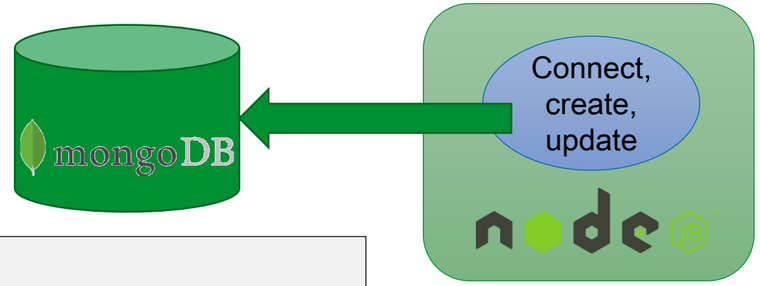
CONNECTING TO SERVER




```
var MongoClient = require('mongodb').MongoClient;

var mongodbHost = '127.0.0.1';
var mongodbPort = '27017';
var mongodbDatabase = 'world';

var url = 'mongodb://'+mongodbHost+':'+mongodbPort + '/' + mongodbDatabase;

MongoClient.connect(url, function(err, db) {
   console.log("Connected correctly to server.");

   // DO YOUR THING WITH MONGODB

   db.close();
   console.log("Connection to database is closed.");
}) //connect()
```

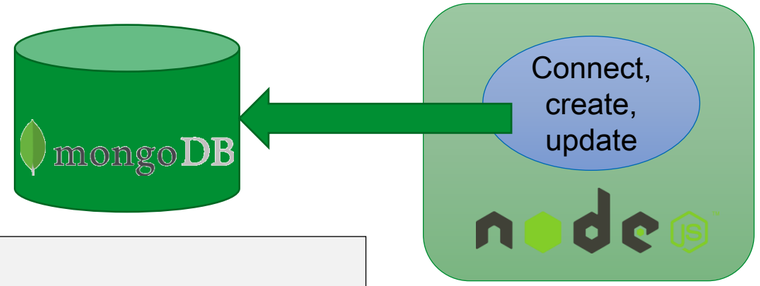
AMIS

RETRIEVE DATA (1)



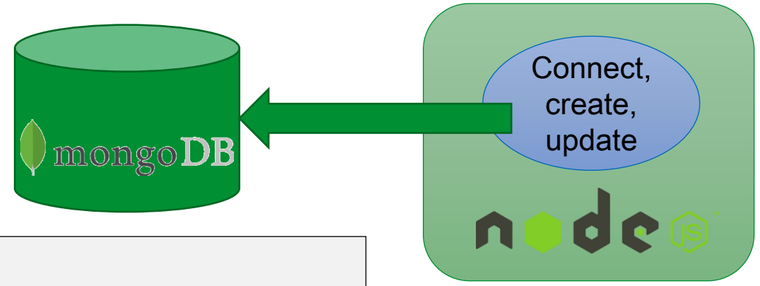

```
var MongoClient = require('mongodb').MongoClient;
var mongodbHost = '127.0.0.1';
var mongodbPort = '27017';
var mongodbDatabase = 'world';
var url = 'mongodb://'+mongodbHost+':'+mongodbPort + '/' + mongodbDatabase;

MongoClient.connect(url, function(err, db) {
   console.log("Connected correctly to server.");
   db.collection('countries').find({},{"sort": [["area",-1]]})
                          .limit(20).toArray(function(err, results){
     console.log("Name of Country Four " +results[3].name+ " and size: " +results[3].area);

     // using cursors to retrieve data sets in a controlled fashion
     // note: cursor implements NodeJS Stream  - results can be piped
     var cursor = db.collection('countries').find({"continent":"Asia"}, {"sort": "name"});

     cursor.count(function(err, count){
       console.log("Country count in Asia: "+count);
     });

     cursor.each(function(err, country){
       console.log(country.name);
     })//each cursor
```

RETRIEVE DATA (2)



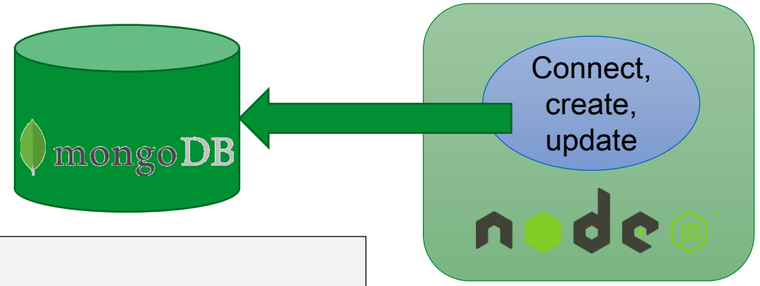

```
var aggquery = [  {$sort: {area : -1}}
                , {$group:{ _id: '$continent'
                           , largestCountry : {$first: "$name"}
                           }
                  }
    ];
    var aggcursor = db.collection('countries').aggregate(aggquery);
    aggcursor.each(function(err, result){
      if (err) {
         console.log(err);
      } else if (result)
         console.log(JSON.stringify(result));
    })  //each aggcursor


var ccursor = db.collection('countries').find({});
// the cursor returned from find and aggregate implements a NodeJS (readable) Stream
ccursor.on('data', function(doc) {
  console.log(doc);
});
ccursor.once('end', function() {
  console.log("Out of countries. Time to move on");
});
```

CONNECTING TO SERVER – ECMA 6



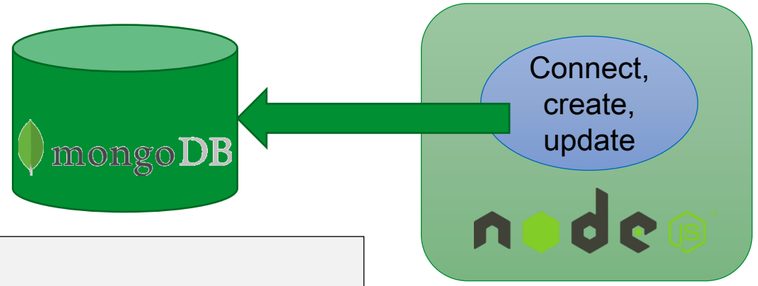

```
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert'),   co = require('co');

var mongodbHost = '127.0.0.1';
var mongodbPort = '27017';
var mongodbDatabase = 'world';
var url = 'mongodb://'+mongodbHost+':'+mongodbPort + '/' + mongodbDatabase;

co(function*() {
  // Use connect method to connect to the Server
  var db = yield MongoClient.connect(url);

  console.log("Connected correctly to server");
  // DO YOUR THING WITH MONGODB

  // Close the connection
  db.close();
}).catch(function(err) {
  console.log(err.stack);
});
```

RETRIEVE DOCUMENTS – ECMA 6



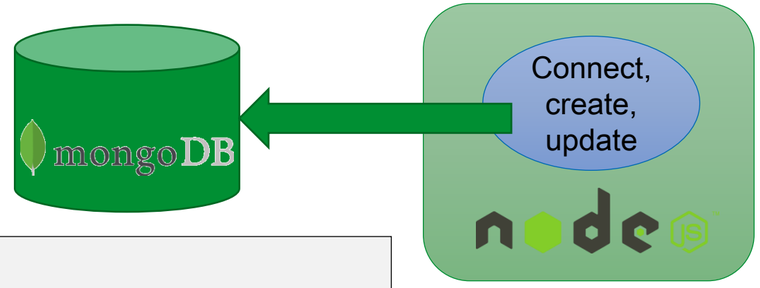

```
co(function*() {
  // Use connect method to connect to the Server
  var db = yield MongoClient.connect(url);

  console.log("Connected correctly to server");
  // find top 20 countries by  size
  var results = yield db.collection('countries').find({}
                          ,{"sort": [["area",-1]]}).limit(20).toArray();
  console.log("Country One " +JSON.stringify(results[0]));
  console.log("Name of Country Four " +results[3].name
                                      + " and size: " +results[3].area);
  // use cursor to get the country count
  var cursor = db.collection('countries').find({"continent":"Asia"}
                                                , {"sort": "name"});
  var count = yield cursor.count();
  console.log("Country count in Asia: "+count);

  while (yield cursor.hasNext()){
    var country = yield cursor.next();
    console.log(country.name);
  }
```

RETRIEVE DOCUMENTS (2) – ECMA 6



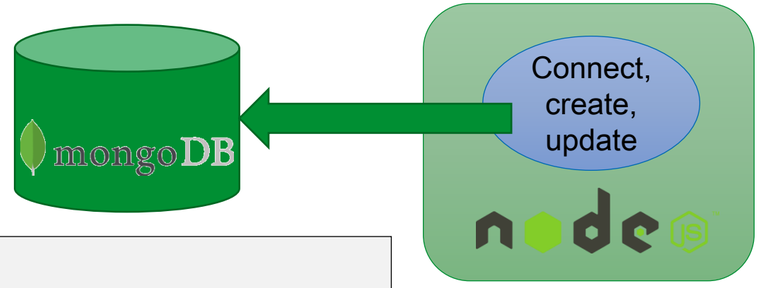

```
// the largest country per continent
    var aggquery = [  {$sort: {area : -1}}
                    ,  {$group:{ _id: '$continent'
                                , largestCountry : {$first: "$name"}
                                }
                       }
     ];
     var aggcursor = db.collection('countries').aggregate(aggquery);
  while (yield aggcursor.hasNext()){
       var result = yield aggcursor.next();
          console.log(JSON.stringify(result));
    }
  // Close the connection
  db.close();
}).catch(function(err) {
  console.log(err.stack);
});//co
```

CREATE DOCS IN MONGODB – ES6



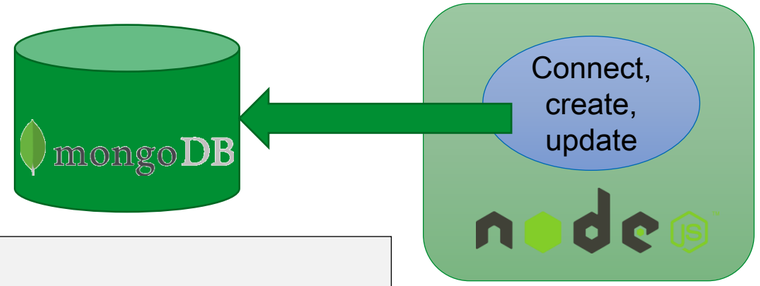

```
// define two documents, any structure you like
  var doc = {
       "continent" : "Europe",
        "children" : [ {"name":"Belgium"}, {"name":"Luxemburg"}],
        "someVariable" : 19123, "andmore" : "2716NK" };
  var doc2 = {
       "continent" : "Asia", "name" : "Johnny",
        "nrs" : [ {"name":"China"}, {"name":"India"}, {"name":"Buthan"}],
       "tree": {"branch": {"twig":{"leaf": 1}}}
     };
  var nameOfCollection = "myDocs"
  var result = yield db.collection(nameOfCollection).insertMany([doc,doc2]);
  console.log(">> "+result.insertedCount
                  +" documents created into collection "+nameOfCollection);
  ...
```

CREATE & UPDATE DOCS – ES6



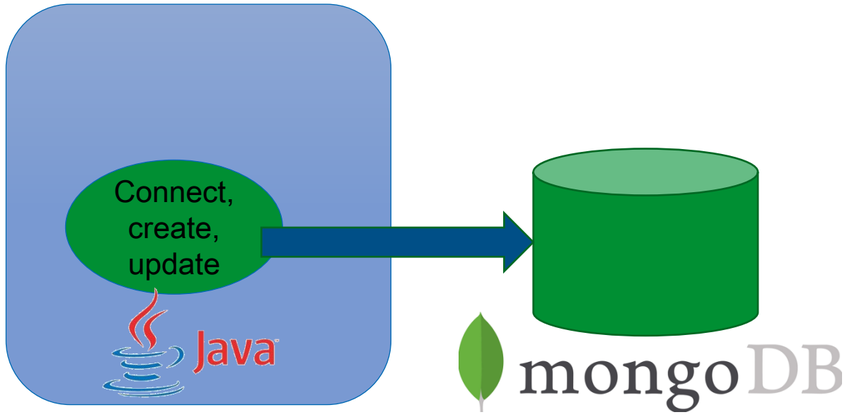

```
// define two documents, any structure you like
   var doc = {...}, doc2 = {...} , nameOfCollection = "myDocs“;
   var result = yield db.collection(nameOfCollection).insertMany([doc,doc2]);
   console.log(">> "+result.insertedCount +" documents created);
   var cursor = db.collection(nameOfCollection).find();
   while (yield cursor.hasNext()){
       var doc  = yield cursor.next();
       console.log("Document: " +JSON.stringify(doc));
    }// while cursor
   result = yield db.collection(nameOfCollection).updateOne(
    {"tree.branch.twig.leaf":1}, {$set: {name: "Hank", city:"Nieuwegein“
                                  , "location.province":"Utrecht“
                                  , "location.country":"The Netherlands“
                                  , "tree.stem":5}});
    console.log(">> Updated "+result.modifiedCount+" document(s)");
    cursor = db.collection(nameOfCollection).find();
    while (yield cursor.hasNext()){
       var doc  = yield cursor.next();
       console.log("Document: " +JSON.stringify(doc));
    }// while cursor
```

CREATE, UPDATE AND DELETE – ES6




```
// define two documents, any structure you like
var doc = {...}, doc2 = {...} , nameOfCollection = "myDocs“;
var result = yield db.collection(nameOfCollection).insertMany([doc,doc2]);
console.log(">> "+result.insertedCount +" documents created);
var cursor = db.collection(nameOfCollection).find();
while (yield cursor.hasNext()){
    var doc  = yield cursor.next();
    console.log("Document: " +JSON.stringify(doc));
 }// while cursor
result = yield db.collection(nameOfCollection).updateOne(...);
...
result = yield db.collection(nameOfCollection).deleteMany({});
console.log(">> Deleted "+ result.deletedCount+" documents ");
// execute command to drop the collection
yield db.command({drop:nameOfCollection});
console.log(">> Dropped collection "+nameOfCollection);
// Close the connection
db.close();
console.log("Connection to database is closed.");
```

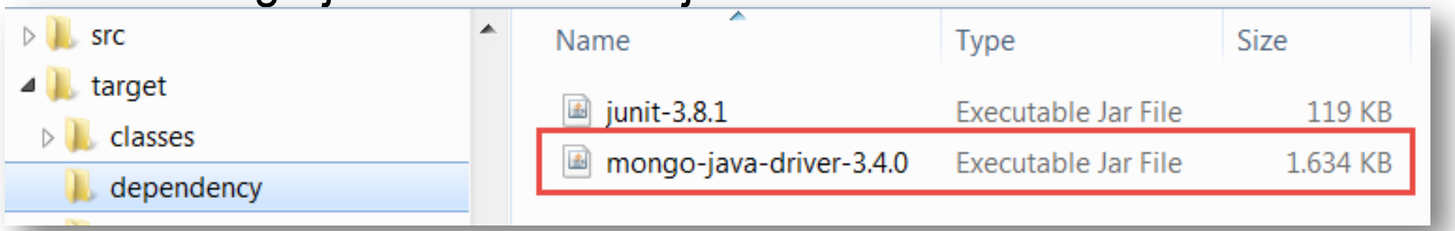
INTERACTING WITH MONGODB FROM JAVA



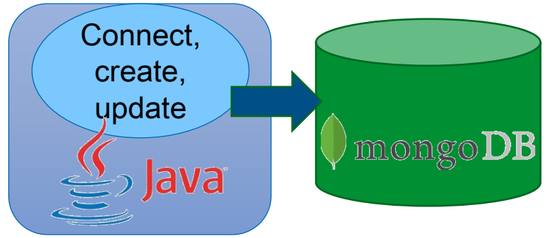
- MongoDB Java Driver 3.x
  - Supports Synchronous and Asynchronous operations
  - https://docs.mongodb.com/ecosystem/drivers/java/
- Using MongoDB Driver in a Java Application
  - Add dependency in Maven pom.xml file

```
<dependencies>
  <dependency>
      <groupId>org.mongodb</groupId>
      <artifactId>mongo-java-driver</artifactId>
      <version>3.4.0</version>
  </dependency>
```

  - Library JAR file mongo-java-driver-3.4.0.jar

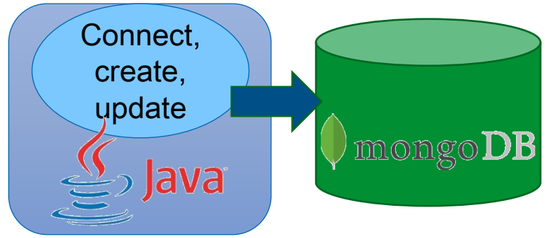


CONNECTING TO SERVER



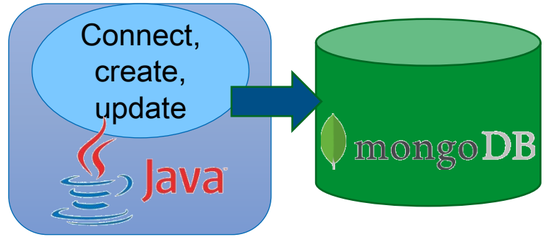

```
package nl.amis.mongodb.countries;

import java.net.UnknownHostException;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

public class AppConnect {
    public static void main(String[] args) throws UnknownHostException {
        String mongodbHost = "127.0.0.1";
        Integer mongodbPort = 27017;
        String mongodbDatabase = "world";

        MongoClient mongoClient = new MongoClient(mongodbHost, mongodbPort);
        try {
            System.out.println("Connected to server; now hook into database " + mongodbDatabase);
            MongoDatabase db = mongoClient.getDatabase(mongodbDatabase);
            System.out.println("Connected to database " + mongodbDatabase + " successfully");
        } finally {
            mongoClient.close();
            System.out.println("Closed mongodb connection");
        }
    }
}
```

RETRIEVE DATA



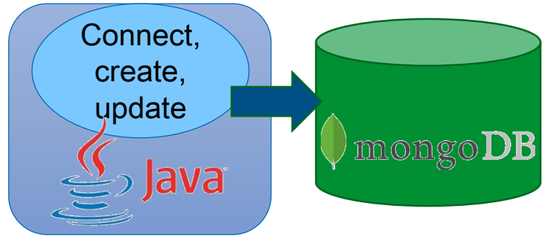

```
...
import org.bson.Document;
import com.mongodb.client.MongoCollection;

public class AppConnect {
    public static void main(String[] args) throws UnknownHostException {
        ... Connect to world database

// get all documents from the countries collection
MongoCollection<Document> countries = db.getCollection("countries");
System.out.println("Got collection countries  successfully");

System.out.println("Number of countries: " + countries.count());

System.out.println("All countries - name and continent");
for (Document doc : countries.find()) {
  // System.out.println(doc.toJson());
  System.out.println(doc.get("name") + " (" + doc.get("continent") + ")");
}//for
```

RETRIEVE DATA (2)



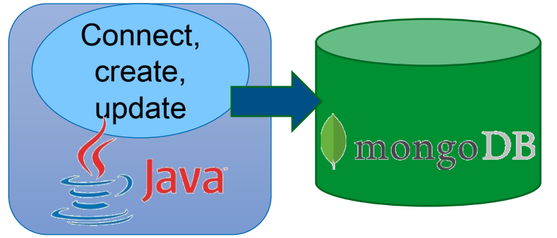

```
import com.mongodb.client.model.Sorts;
import com.mongodb.client.model.Filters;
import com.mongodb.Block;

public class AppConnect {
    public static void main(String[] args) throws UnknownHostException {
        ... Connect to world database

System.out.println( "Retrieve all countries in Asia order alphabetically
                            by name  and display full country document");

countries.find(Filters.eq("continent", "Asia"))
         .sort(Sorts.ascending("name"))
         .forEach(new Block<Document>() {
                    @Override
                    public void apply(final Document document) {
                      System.out.println(document.toJson());
                    }
          });
```

RETRIEVE DATA - AGGREGATION



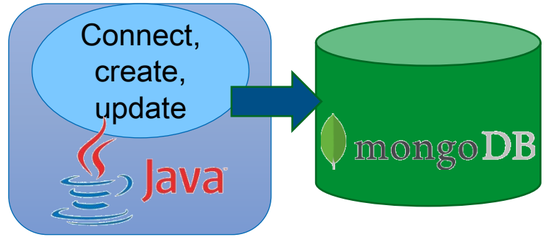

```
...
Block<Document> printBlock = new Block<Document>() {
        @Override
        public void apply(final Document document) {
                System.out.println("Country: " + document.toJson());
        }
};

...
System.out.println("List largest country in each continent");
countries.aggregate(
   Arrays.asList(
      Aggregates.sort(Sorts.descending("area")),
      Aggregates.group("$continent", Accumulators.first("Largest Country", "$name"))
  ))
  .forEach(printBlock);
```

RETRIEVE DATA – AGGREGATION (2)



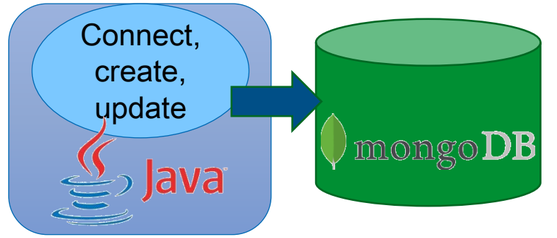

```
System.out.println("List name in uppercase,sorted by size, delta between birthdate and deathrate, "
                          +" population density for countries in Oceania");

BsonArray subArgs=new BsonArray();
            subArgs.add(new BsonString("$birthrate"));
            subArgs.add(new BsonString("$deathrate"));
BsonArray divArgs=new BsonArray();
            divArgs.add(new BsonString("$population"));
            divArgs.add(new BsonString("$area"));

countries.aggregate
  (Arrays.asList(
     Aggregates.match(Filters.eq("continent", "Oceania")),
     Aggregates.sort(Sorts.descending("area")),
     Aggregates.project( Projections.fields(
         Projections.excludeId(),
         Projections.include("continent"),
         Projections.computed("name",Projections.computed("$toUpper",  "$name")),
         Projections.computed("populationGrowthRate",Projections.computed("$subtract",  subArgs)),
         Projections.computed("populationDensity",
                         Projections.computed("$trunc", Projections.computed("$divide",  divArgs)))
     ))
  ))
  .forEach(printBlock);
```

JAVA: CREATE DOCUMENTS IN MONGODB



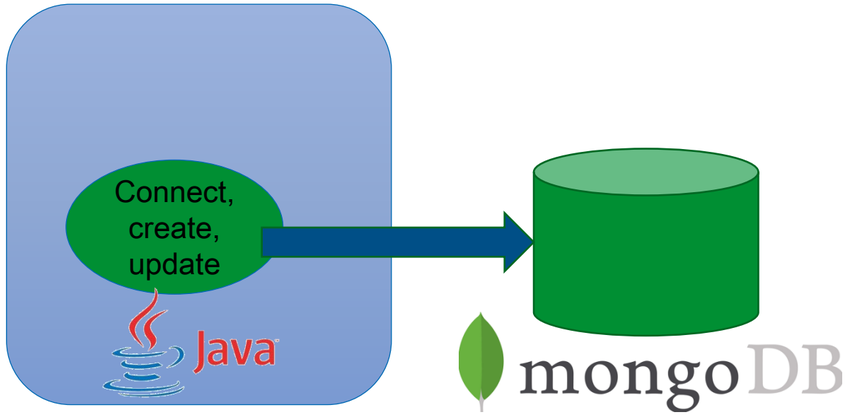

```
...
String nameOfCollection = "myDocs";
MongoCollection<Document> coll = db.getCollection(nameOfCollection);
Document doc1 = new Document("continent", "Europe")
                .append("nrs", Arrays.asList(new Document("name", "Belgium"),
        new Document("name", "Luxemburg")))
                .append("someVariable", 19123)
                .append("andmore", "kl;jdsfakhfsdahjfsdasjbsdahjbsdahgvsahjkZl;po");
Document doc2 = new Document("continent", "Asia").append("name", "Johnny")
                .append("nrs", Arrays.asList(new Document("name", "China"),
        new Document("name", "India"), new Document("name", "Buthan")))
                .append("tree", new Document("branch",
        new Document("twig", new Document("leaf", 1))));
List<Document> documents = new ArrayList<Document>();
documents.add(doc1);
documents.add(doc2);

coll.insertMany(documents);
System.out.println(">> Documents created into collection");
coll.find().forEach(printBlock);
```

JAVA: UPDATE AND DELETE DATA




```
...
coll.updateOne(
    Filters.eq("tree.branch.twig.leaf", 1),
    Updates.combine(
      Updates.set("name", "Hank"), Updates.set("city", "Nieuwegein"),
      Updates.set("location.province", "Utrecht"),
      Updates.set("location.country", "The Netherlands"),
      Updates.set("tree.stemp", 5),
      Updates.currentDate("lastModified")
    )
);
System.out.println(">> Updated one document in collection " + nameOfCollection);
coll.find().forEach(printBlock);

DeleteResult dr = coll.deleteMany(Filters.exists("_id"));
System.out.println(">> Deleted " + dr.getDeletedCount() + " documents from collection "
                + nameOfCollection);
db.runCommand(new Document("drop", nameOfCollection));
System.out.println(">> Dropped  collection " + nameOfCollection);
```

INTERACTING WITH MONGODB FROM JAVA – JSON ⇔ OO MAPPING



- Libraries for mapping between BSON/JSON documents and Java Objects
  - Morphia - http://mongodb.github.io/morphia/1.3/getting-started/quick-tour/
  - Jongo – Query in Java as in the MongoDB shell - http://jongo.org/

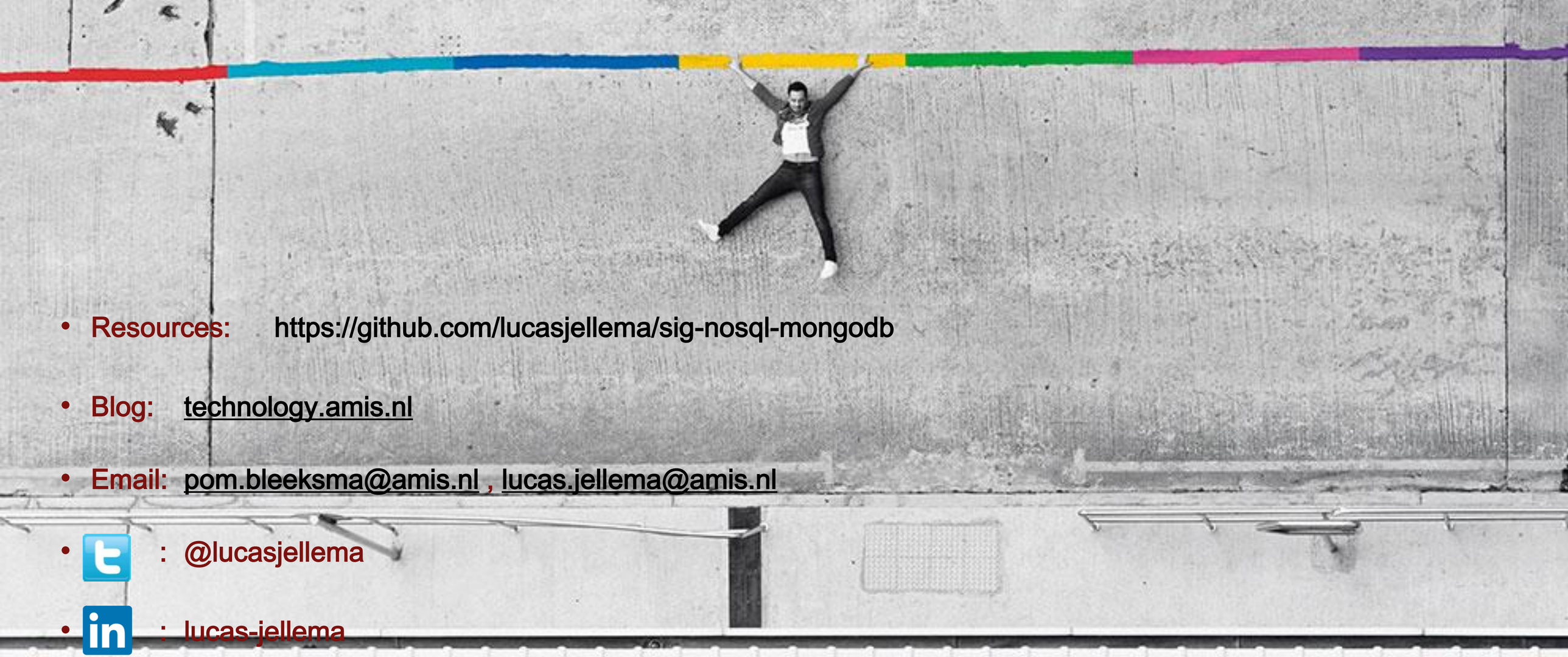
```
Jongo jongo = new Jongo(db);
MongoCollection friends = jongo.getCollection("friends");

MongoCursor<Friend> all = friends.find("{name: 'Joe'}").as(Friend.class);
Friend one = friends.findOne("{name: 'Joe'}").as(Friend.class);
```

AMIS

HANDS ON WORKSHOP

- Install MongoDB
- Create Database, Create Collections and Documents
- Retrieve Data
- Import Database
- More configuration and administration
- Development: interacting with MongoDB from
  - Compass
  - Java
  - Node.js

AMIS



- Resources: https://github.com/lucasjellema/sig-nosql-mongodb
- Blog: technology.amis.nl
- Email: pom.bleeksma@amis.nl , lucas.jellema@amis.nl
- : @lucasjellema
- : lucas-jellema
- : www.amis.nl, info@amis.nl
  +31 306016000
  Edisonbaan 15,
  Nieuwegein