Document Databases

UA.DETI.CBD
José Luis Oliveira / Carlos Costa

Outline

- Document databases
  - General introduction
  - Relational versus Document stores

- MongoDB
  - Data model
  - CRUD operations
    - Insert, Update, Remove
    - Find: projection, selection, modifiers
  - Index structures

- Java driver

Storage example: *Linkedin* in RDMS

http://www.linkedin.com/in/williamhgates

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

users table

user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of … blogger.

region_id	industry_id	photo_id
us:91	131	57817532

regions table

id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table

id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table

id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F…
457	251	Co-founder, Chairman	Microsoft

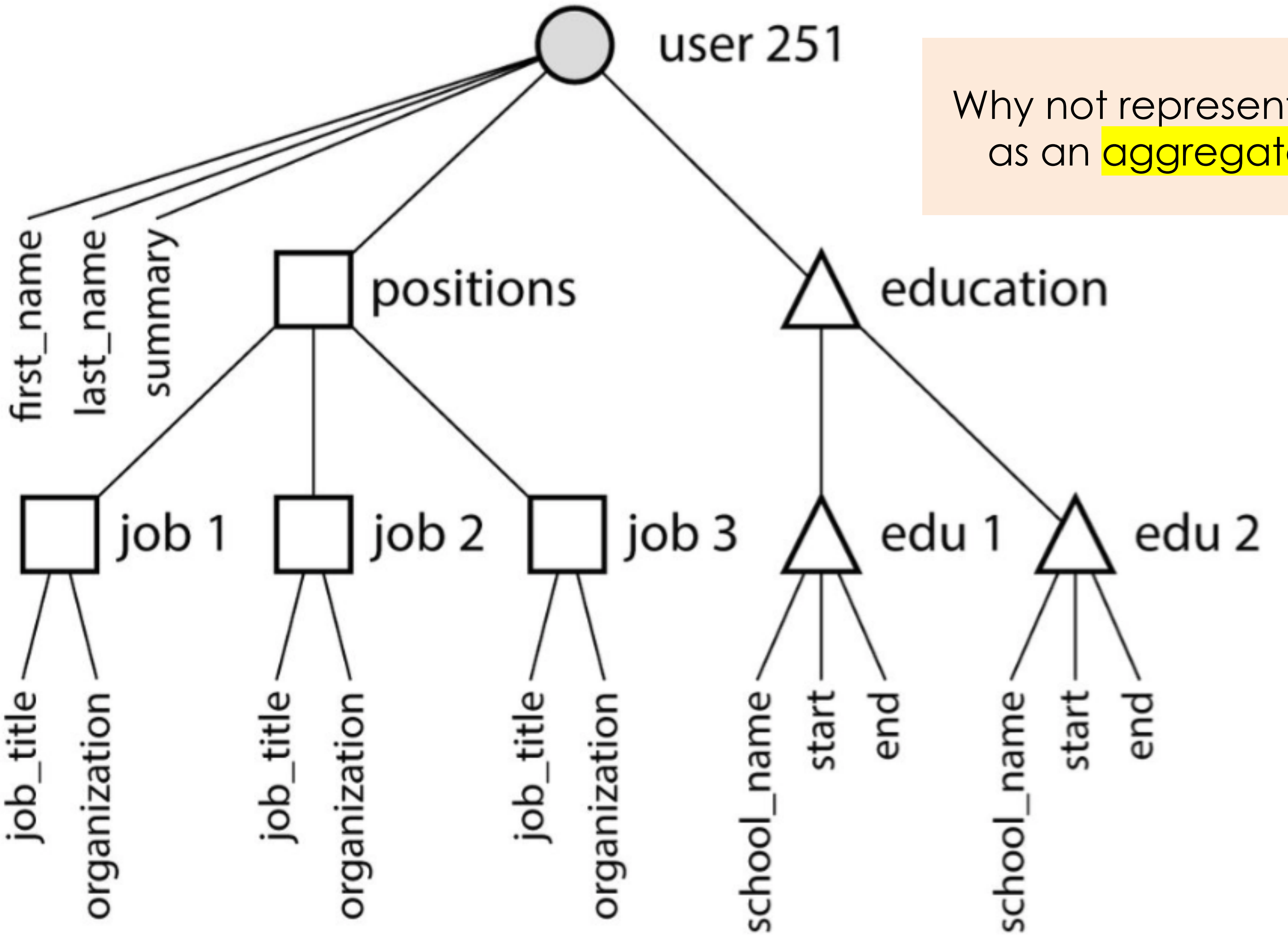
education table

id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

contact_info table

id	user_id	type	url
155	251	blog	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates

One-to-Many relations



Why not represent this as an aggregate?

JSON representation


```
{
    "user_id": 251,
    "first_name": "Bill",
    "last_name": "Gates",
    "summary": "Co-chair of the Bill & Melinda Gates... Active blogger.",
    "region_id": "us:91",
    "photo_url": "/p/7/000/253/05b/308dd6e.jpg",
    "positions": [
        {"job_title": "Co-chair", "organization": "B&M Gates Foundation"},
        {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
    ],
    "education": [
        {"school_name": "Harvard University", "start": 1973, "end": 1975},
        {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
    ]
}
```

Document vs. Relational databases

- The main arguments in favour of the document data model are:
  - simpler application code, schema flexibility, and better performance due to locality.

- May be used for data with a document-like structure,
  - i.e. a tree of one-to-many relationships, where typically the entire tree is loaded at once.
  - When splitting a document-like structure into multiple tables can lead to unnecessarily complicated application code.
    - Event logging, content management systems, blogs, web analytics, e-commerce applications, …

Documents for schema flexibility

- The *schemaless* approach is **advantageous if the data is heterogeneous**
  - i.e. the items in the collection don't all have the same structure.
- For example, because:
  - there are many different types of objects, and it is not practical to put each type of object in its own table, or
  - Data structure determined by external systems, over which we have no control, and which may change at any time.
- In situations like these, a schema may hurt more

Documents for schema flexibility

- A document is usually stored as a **single continuous string**, encoded as JSON, XML or a binary variant thereof (such as MongoDB's BSON).
  - If one needs to access the entire document, there is a performance advantage to this storage locality.
- The **locality advantage** only applies if you need large parts of the document at the same time.
  - The database typically needs to load the entire document, even if you access only a small portion of it, which can be wasteful on large documents.
  - On updates to a document, the entire document usually needs to be re-written.
- Recommended to keep documents small.

Document vs. Relational databases

- When not to use Document
  - Set operations involving multiple documents
  - Design of document structure is constantly changing
    - i.e. when the required level of granularity would outbalance the advantages of aggregates
- If the application does use **many-to-many relationships**, the document model becomes less appealing (no joins).
  - We may denormalize the database, or joins can be emulated in application code by making multiple requests to the database.
  - But… the problems of managing denormalization and joins may be greater than the problem of object-relational mismatch.

Convergence of document and relational databases

- Most **relational** database systems also started supporting XML and JSON
  - i.e. functions to create/update documents, and the ability to index and query inside documents.
  - This allows applications to use data models similar to document databases.

- On the **document** database side..
  - Several solutions have also evolving to provide SQL-like experience in document databases (MongoDB Atlas, RethinkDB, Knomi, ObjectRocket, …)

Document Stores

- Data model
  - **Documents**
    - Self-describing
    - Hierarchical tree structures (JSON, XML, …) – Scalar values, maps, lists, sets, nested documents, …
    - Identified by a unique identifier (key, …)
  - **Collections** – a set of documents
- Query patterns (CRUD)
  - **C**reate, **U**pdate or **D**elete a document
  - **R**ead/retrieve documents according to complex query conditions

  - Extended key-value stores where the value part is examinable

Document Stores

- Document
  - MongoDB, Couchbase, CouchDB,
  - RethinkDB, RavenDB,
  - Google Cloud Firestore
- Multi-model
  - MarkLogic, OrientDB, ArangoDB
  - Amazon DynamoDB,
  - Microsoft Azure Cosmos DB,
  - *... many others*

DB-Engines Ranking of Document Stores

Rank			DBMS	Database Model	Score		
Oct 2020	Sep 2020	Oct 2019			Oct 2020	Sep 2020	Oct 2019
1.	1.	1.	MongoDB	Document, Multi-model	448.02	+1.54	+35.93
2.	2.	2.	Amazon DynamoDB	Multi-model	68.41	+2.23	+8.24
3.	3.	↑ 4.	Microsoft Azure Cosmos DB	Multi-model	32.01	+0.34	+0.68
4.	4.	↓ 3.	Couchbase	Document, Multi-model	30.33	-0.28	-1.88
5.	5.	5.	CouchDB	Document	17.41	+0.16	-0.63
6.	6.	↑ 7.	Firebase Realtime Database	Document	16.26	+0.65	+4.52
7.	7.	↓ 6.	MarkLogic	Multi-model	11.73	-0.21	-1.33
8.	8.	8.	Realm	Document	8.74	0.00	+0.82
9.	9.	9.	Google Cloud Firestore	Document	8.61	+0.56	+3.28
10.	↑ 11.	10.	Google Cloud Datastore	Document	5.93	+0.14	+0.67
11.	↓ 10.	↑ 12.	ArangoDB	Multi-model	5.55	-0.25	+0.67
12.	12.	↓ 11.	OrientDB	Multi-model	5.47	-0.01	+0.34

https://db-engines.com/en/ranking/document+store

MongoDB Document Database

mongoDB®

MongoDB

- JSON document database
  - https://www.mongodb.com/
- Features
  - Open source, high availability, eventual consistency, automatic sharding, master-slave replication, automatic failover, secondary indexes, …
- Developed by MongoDB
- Implemented in C++, C, and JavaScript
- Operating systems: Windows, Linux, Mac OS X, …
- Initial release in 2009

Data Model

- Structure
  - Instance → databases → collections → documents

- **Database**
  - Set of Collections
- **Collection**
  - Set of Documents, usually of a similar structure
- **Document**
  - MongoDB document = one JSON object
  - Internally stored as BSON
  - Each document…
    - belongs to exactly one collection
    - has a unique identifier `_id`

```
{
  name: "martin",
  age: 22,
  interests: [ sports, CBD ]
}
```

UNIVERSIDADE DE AVEIRO

Example

- Collection `redwine`

```
{
  _id: "1",
  name: "Cartuxa",
  year: 2012
}
{
  _id: "2",
  name: "Evel",
  year: 2010
}
{
  _id: "3",
  name: ”EA",
  year: 2016
}
```

- Query statement

Wines older than 2012 and later, sorted by these titles in descending order

```
db.redwine.find(
{ year: { $lt: 2014} },
{ _id: false, name: true } )
.sort({ name: -1 })
```

- Query result

```
{ "name" : "Evel" }
{ "name" : "Cartuxa" }
```

Data Model – Primary Keys

- **_id** is reserved for a primary key
  - Unique within a collection
  - Immutable (cannot be changed once assigned)
  - Can be of any type other than an array
- Possible values
  - Natural identifier (e.g. a key)
    - Must be unique!
  - UUID (Universally unique identifier)
    - 16-byte number (ISO/IEC 11578:1996, RFC 4122)
  - **ObjectId**
    - Special 12-byte BSON type (default option)
    - Small, likely unique, fast to generate, ordered, based on a timestamp, machine id, process id, and a process-local counter

Data Model – Denormalized

- **Embedded documents**
  - Related data in a single structure with subdocuments
  - Suitable for one-to-one or one-to-many relationships
  - Brings ability to read / write related data in a single operation
    - i.e. better performance, less queries need to be issued

```
> db.redwine.insert( {
   winepack: "Dinner",
   bottles: [
      { name: "Cartuxa", year: 2012 },
      { name: "Evel", year: 2010 },
      { name: "EA", year: 2016 }
      ]
   })
```

Data Model – Normalized

❖ References

- Directed links between documents, expressed via identifiers
  - Idea analogous to foreign keys in relational databases
  - Suitable for **many-to-many relationships**
  - Embedding in this case would result in data duplication
- References provide more flexibility than embedding
  - But follow up queries are needed

```
> db.redwine.insert( {
   winepack: "Dinner",
   bottles: [
       { "$id" : "1" },
       { "$id" : "3" }
       ]
   })
```

```
{ _id: "1",
   name: "Cartuxa",
   year: 2012 }
{ _id: "2",
   name: "Evel",
   year: 2010 }
{ _id: "3",
   name: ”EA",
   year: 2016 }
```

  - The $id field contains the value of the _id field in the referenced document.

Tools

- MongoDB Atlas – remove server
  - https://www.mongodb.com/cloud/atlas
- Local installation
  - https://www.mongodb.com/try/download/community
  - Local server

```
$ mongod --dbpath <path to data directory>
```

- Mongo client
  - interactive JavaScript interface to MongoDB.

```
$ mongo
$ mongo --username user --password pass --host host --port 28015
```

- Other tools
  - bsondump, dump, mongodump, mongoexport, mongofiles, mongoimport, mongooplog, mongoperf, mongoreplay, mongorestore, mongos, mongostat, mongotop

Query Language

- JavaScript commands
  - Each individual command is evaluated over exactly one collection
  - Queries return a cursor
    - Allows us to iterate over all the selected documents

- Query patterns
  - Basic CRUD operations
    - Accessing documents via identifiers or conditions on fields
  - Aggregations: MapReduce, pipelines, grouping

CRUD Operations

- Create
  - db.collection.**insertOne**()
  - db.collection.**insertMany**()
- Read
  - db.collection.**find**()
    - Finds documents based on filtering/projection/sorting conditions
- Update
  - db.collection.**updateOne**()
  - db.collection.**updateMany**()
- Delete
  - db.collection.**deleteOne**()
  - db.collection.**deleteMany**()

https://docs.mongodb.com/manual/crud/

Create – insert examples

```
> db.invoice.insertOne({ _id: 901, inv_no: "I001", inv_date: "20171010" })
{ "acknowledged" : true, "insertedId" : 901 }

> db.orders.insertMany(
...     [
...       { _id: 15, ord_no: 2001, qty: 200, unit: "doz" },
...       { ord_no: 2005, qty: 320 },
...       { ord_no: 2008, qty: 250, rate:85 }
...     ]
... );
{
    "acknowledged" : true,
    "insertedIds" : [
        15,
        ObjectId("59b1a6d6935c2a0ca72c432a"),
        ObjectId("59b1a6d6935c2a0ca72c432b")
    ]
}
```

UNIVERSIDADE DE AVEIRO

Read/query operation

```
> db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);

> db.inventory.find( {} )  // SELECT * FROM inventory
{ "_id" : ObjectId("59b1b730935c2a0ca72c432c"), "item" : "journal", "qty" : 25,
"size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("59b1b730935c2a0ca72c432d"), "item" : "notebook", "qty" : 50,
"size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("59b1b730935c2a0ca72c432e"), "item" : "paper", "qty" : 100,
"size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("59b1b730935c2a0ca72c432f"), "item" : "planner", "qty" : 75,
"size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
{ "_id" : ObjectId("59b1b730935c2a0ca72c4330"), "item" : "postcard", "qty" : 45,
"size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
```

Selection

```
> db.inventory.find( { status: "D" } )
                    // SELECT * FROM inventory WHERE status = "D"

> db.inventory.find( { status: { $in: [ "A", "D" ] } } )
                    // SELECT * FROM inventory WHERE status in ("A", "D")

> db.inventory.find( { status: "A", qty: { $lt: 30 } } )
                    // SELECT * FROM inventory WHERE status = "A" AND qty < 30

> db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
                    // SELECT * FROM inventory WHERE status = "A" OR qty < 30

> db.inventory.find( {
    status: "A",
    $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
} )     // SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

Selection operators

- Comparison
  - **`$eq, $ne`**
    - Tests the actual field value for equality / inequality
  - **`$lt, $lte, $gte, $gt`**
    - Less than / less than or equal / greater than or equal / greater
  - **`$in`**
    - Equal to at least one of the provided values
  - **`$nin`**
    - Negation of `$in`
- Logical
  - **`$and, $or`**
  - **`$nor`**
    - returns all documents that fail to match both clauses.
  - **`$not`**

Selection operators

- Element operators
  - `$exists`
    - tests whether a given field exists / not exists
  - `$type`
    - selects documents if a field is of the specified type.

- Evaluation operators
  - `$regex`
    - tests whether the field value matches a regular expression (PCRE)
  - `$text`
    - performs text search (text index must exists)

Selection operators

- Array query operators
  - **`$all`**
    - Matches arrays that contain all elements specified in the query.
  - **`$elemMatch`**
    - Selects documents if an element in the array field matches all the specified $elemMatch conditions.
  - **`$size`**
    - Selects documents if the array field is a specified size.

Projection

```
// SELECT _id, item, status FROM inventory
> db.inventory.find( { } , { item: 1, status: 1 } )
{ "_id" : ObjectId("59b1bd23ed835ca4380da8b2"), "item" : "journal", "status" : "A" }
{ "_id" : ObjectId("59b1bd23ed835ca4380da8b3"), "item" : "notebook", "status" : "A" }
{ "_id" : ObjectId("59b1bd23ed835ca4380da8b4"), "item" : "paper", "status" : "D" }
{ "_id" : ObjectId("59b1bd23ed835ca4380da8b5"), "item" : "planner", "status" : "D" }
{ "_id" : ObjectId("59b1bd23ed835ca4380da8b6"), "item" : "postcard", "status" : "A" }

// SELECT item, status FROM inventory
> db.inventory.find( { } , { _id: 0, item: 1, status: 1 } ) // true or 1 is included
{ "item" : "journal", "status" : "A" }
{ "item" : "notebook", "status" : "A" }
{ "item" : "paper", "status" : "D" }
{ "item" : "planner", "status" : "D" }
{ "item" : "postcard", "status" : "A" }

> db.inventory.find( {} , { _id: 0, qty: 0, size: 0 } ) // false or 0 is excluded
{ "item" : "journal", "status" : "A" }
{ "item" : "notebook", "status" : "A" }
{ "item" : "paper", "status" : "D" }
{ "item" : "planner", "status" : "D" }
{ "item" : "postcard", "status" : "A" }
```

Modifiers (sort, limit, skip)

```
// SELECT _id, item, status FROM inventory ORDER BY status ASC
> db.inventory.find( {} , { _id: 0, item: 1, status:1 }).sort({ status: 1 })
{ "item" : "journal", "status" : "A" }
{ "item" : "notebook", "status" : "A" }
{ "item" : "postcard", "status" : "A" }
{ "item" : "paper", "status" : "D" }
{ "item" : "planner", "status" : "D" }

> db.inventory.find( {} , { _id: 0, item: 1, status:1 }).sort({ status: -1 })
{ "item" : "paper", "status" : "D" }
{ "item" : "planner", "status" : "D" }
{ "item" : "journal", "status" : "A" }
{ "item" : "notebook", "status" : "A" }
{ "item" : "postcard", "status" : "A" }
> db.inventory.find( {} , { _id: 0, item: 1, status:1 }).limit(3)
{ "item" : "journal", "status" : "A" }
{ "item" : "notebook", "status" : "A" }
{ "item" : "paper", "status" : "D" }
> db.inventory.find( {} , { _id: 0, item: 1, status:1 }).skip(3)
{ "item" : "planner", "status" : "D" }
{ "item" : "postcard", "status" : "A" }
```

CRUD Operations – Update

- Syntax

```
db.collection.updateOne(filter, update, options)
db.collection.updateMany(filter, update, options)

db.collection.updateOne(
     <filter>,   // = selectors in find()
     <update>,   // modification to apply
     {             // optional …
           upsert: <boolean>,         // if no doc -> insert
           writeConcern: <document>, // ack of num of replicas
           collation: <document>     // language/type-specific rules
           ..
     }
)
```

- Update operators

```
     $set, $unset, $rename
```

https://docs.mongodb.com/manual/reference/delete-methods/

Update

```
> db.inventory.find({"item":"journal"}, {_id:0, size:0})
{ "item" : "journal", "qty" : 25, "status" : "A" }

> db.inventory.updateOne({"item":"journal"}, {$set: {"status":"B"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.inventory.find({"item":"journal"}, {_id:0, size:0})
{ "item" : "journal", "qty" : 25, "status" : "B" }

> db.inventory.updateOne({"item":"computer"},
    {$set: {"status":"C", qty:30 } },
    {upsert:true})
{ "acknowledged" : true, "matchedCount" : 0, "modifiedCount" : 0,
"upsertedId" : ObjectId("59b2524f92403315277cbd8f") }

> db.inventory.find( {"item":"computer"} )
{ "_id" : ObjectId("59b2524f92403315277cbd8f"), "item" : "computer",
"status" : "C", "qty" : 30 }
```

Update

```
> db.inventory.updateMany({}, {$unset: { size:""}})
{ "acknowledged" : true, "matchedCount" : 5, "modifiedCount" : 5 }

> db.inventory.find()
{ "_id" : ObjectId("59b1b730935c2a0ca72c432c"), "item" : "journal", "qty" :
25, "status" : "A" }
{ "_id" : ObjectId("59b1b730935c2a0ca72c432d"), "item" : "notebook", "qty"
: 50, "status" : "A" }
{ "_id" : ObjectId("59b1b730935c2a0ca72c432e"), "item" : "paper", "qty" :
100, "status" : "D" }
{ "_id" : ObjectId("59b1b730935c2a0ca72c432f"), "item" : "planner", "qty" :
75, "status" : "D" }
{ "_id" : ObjectId("59b1b730935c2a0ca72c4330"), "item" : "postcard", "qty"
: 45, "status" : "A" }
```

CRUD Operations – Delete

- Syntax

```
db.collection.deleteOne(filter, options)
db.collection.deleteMany(filter, options)

db.collection.deleteOne(
    <filter>,   // = selectors in find()
    {           // optional …
        writeConcern: <document>, // ack of num of replicas
        collation: <document>     // language-specific rules
        ..
    }
)
```

https://docs.mongodb.com/manual/reference/delete-methods/

Delete

```
> db.inventory.find( {} , { _id: 0, qty: 0, size: 0 } )
{ "item" : "journal", "status" : "B" }
{ "item" : "notebook", "status" : "A" }
{ "item" : "paper", "status" : "D" }
{ "item" : "planner", "status" : "D" }
{ "item" : "postcard", "status" : "A" }
{ "item" : "computer", "status" : "C" }

> db.inventory.deleteOne({"item":"computer"})
{ "acknowledged" : true, "deletedCount" : 1 }

> db.inventory.find( {} , { _id: 0, qty: 0, size: 0 } )
{ "item" : "journal", "status" : "B" }
{ "item" : "notebook", "status" : "A" }
{ "item" : "paper", "status" : "D" }
{ "item" : "planner", "status" : "D" }
{ "item" : "postcard", "status" : "A" }
```

Indexes

- **Motivation**
  - Full collection scan must be performed when searching for the documents, unless an appropriate index exists
- **Primary** index
  - MongoDB creates a unique index on the *_id* field during the creation of a collection
- **Secondary** indexes
  - Created manually for a given key field / fields
  - To create an index, use *db.collection.createIndex()* or a similar method from your driver.

```
db.<collection>.createIndex(keys, options)
```

  - MongoDB indexes use a B-tree data structure.

Index Types

- **Single** Field
  - Ascending/descending indexes on a single field.
- **Compound** Index
  - Indexes on multiple fields
    - The order of fields listed in a compound index has significance
    - e.g. { userid: 1, score: -1 }, sort by userid ASC and then, by score DESC.
- **Multikey** Index
  - To index a field that holds an array value.
- **Text** Indexes
- **Hashed** Indexes
- **Geospatial** Index

Index Types

- **1, -1** – standard ascending / descending value indexes
  ```
  db.<collection>.createIndex( { field: -1 } )
  ```
- **hashed** – hash values of a single field are indexed
  ```
  db.<collection>.createIndex( { _id: "hashed" } )
  ```
- **text** – basic full-text index
  ```
  db.<collection>.createIndex( { comments: "text" } )
  ```
- **2d** – points in planar geometry
  ```
  db.<collection>.createIndex( { <location field> : "2d" , <additional
  field> : <value> } , { <index-specification options> } )
  ```
- **2dsphere** – points in spherical geometry
  ```
  db.<collection>.createIndex( { <location field> : "2dsphere" } )
  ```

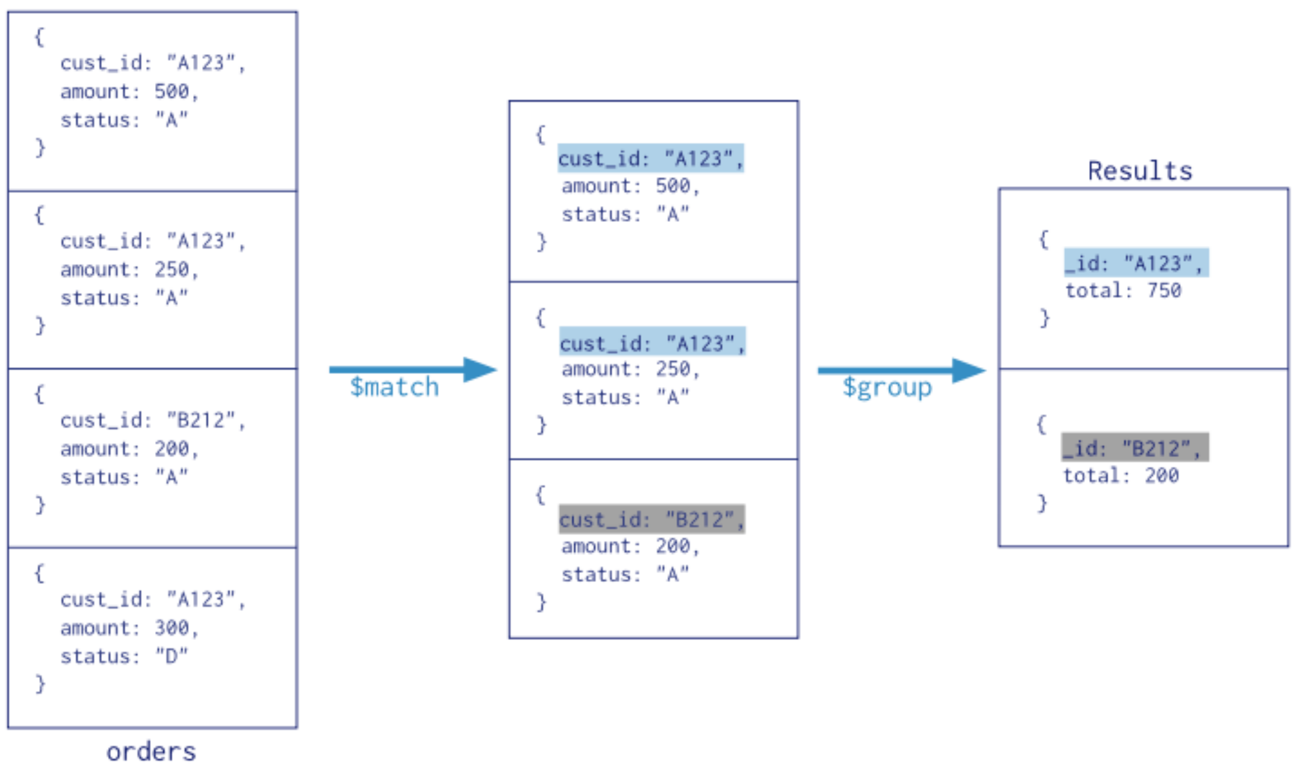
Indexes

```
// Full collection scan
> db.inventory.find( {qty: { "$gte" : 50 }}, {_id:0}).sort( {qty: -1})
{ "item" : "paper", "qty" : 100, "status" : "D" }
{ "item" : "planner", "qty" : 75, "status" : "D" }
{ "item" : "notebook", "qty" : 50, "status" : "A" }

> db.inventory.getIndexes()
[
   {"v": 2, "key": { "_id": 1 }, "name": "_id_", "ns": "test.inventory" }
]

> db.inventory.createIndex( { qty : 1 } )

> db.inventory.getIndexes()
[
   {"v": 2, "key": { "_id": 1 }, "name": "_id_", "ns": "test.inventory" }
   {"v": 2, "key": { "qty": 1 }, "name": "qty_1", "ns": "test.inventory" }
]
```

Aggregation pipeline

– Documents enter a multi-stage pipeline that transforms the documents into aggregated results

```
  Collection
      ↓
db.orders.aggregate( [
     $match stage ──►   { $match: { status: "A" } },
     $group stage ──►   { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
                    ] )
```

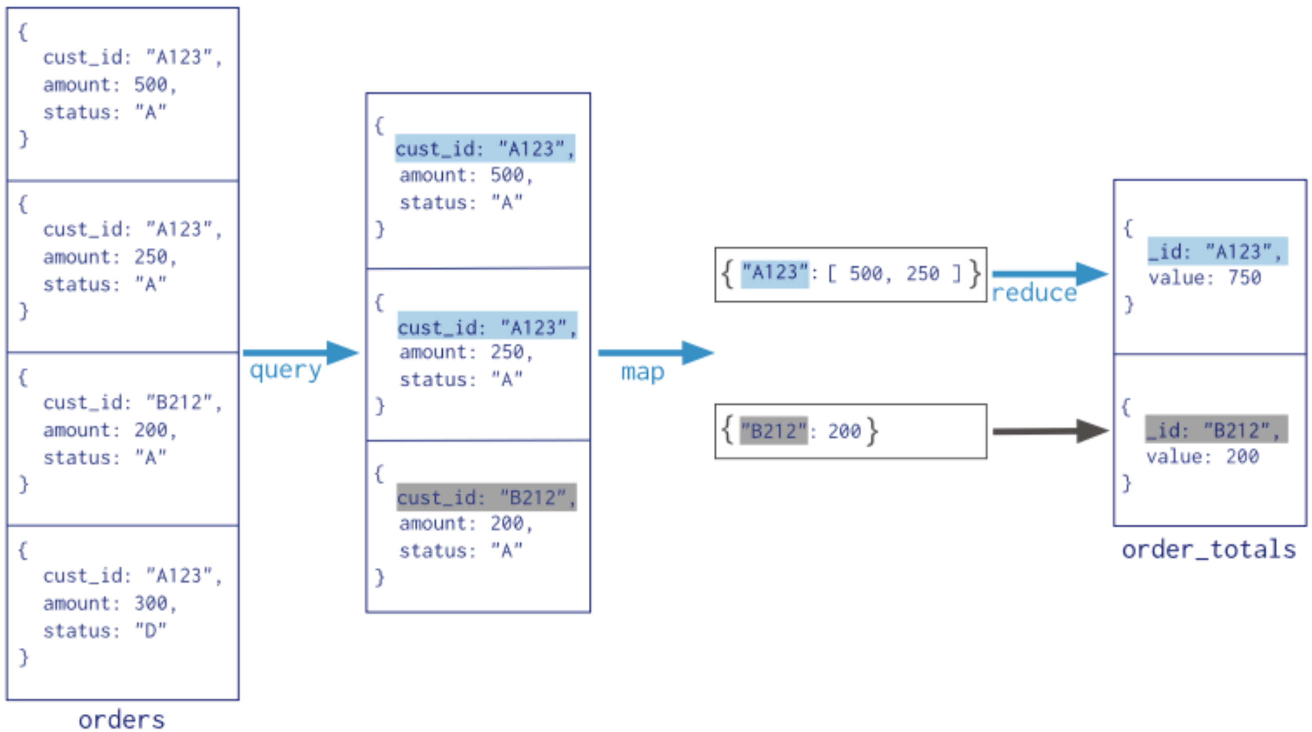


MapReduce

- Data processing paradigm for condensing large volumes of data into useful *aggregated* results.
- Both map and reduce functions are implemented as ordinary JavaScript functions
    - **Map** function: current document is accessible via this, emit(key, value) is used for emissions
    - **Reduce** function: key and array of values are provided as arguments, reduced value is published via return
- Beside others, **query**, **sort** or **limit** options are accepted
    - **out** option determines the output (e.g. a collection name)

MapReduce example

```
Collection
    ↓
db.orders.mapReduce(
       map     →  function() { emit( this.cust_id, this.amount ); },
       reduce  →  function(key, values) { return Array.sum( values ) },
                  {
       query   →    query: { status: "A" },
       output  →    out: "order_totals"
                  }
                )
```



MongoDB Drivers

- The MongoDB Ecosystem contains documentation for the drivers, frameworks, tools, and platform services that work with MongoDB.
  - https://docs.mongodb.com/ecosystem/drivers/
- Drivers are available for many languages
  - C, C++, Java, Python, Ruby, …
- Java
  - http://mongodb.github.io/mongo-java-driver/
    - bson.jar
    - mongodb-driver-core.jar
    - mongodb-driver.jar

Java driver – example 1 (list)

```
public class Test {
    public static void main(String[] args) {
            // remove log in the console
        java.util.logging.Logger.getLogger("org.mongodb.driver").setLevel(
                Level.SEVERE);
        MongoClient mongo = new MongoClient("localhost", 27017);
            // os dados foram colocados manualmente no mongo
        MongoDatabase out = mongo.getDatabase("test");
        System.out.println("-- Colecções na BD " + "'"  + out.getName() + "'" );
        MongoIterable<String> x = out.listCollectionNames();
        for (String s : x)
            System.out.println(s);
        MongoCollection<Document> c = out.getCollection("inventory");
        System.out.println("-- Total de documentos em 'inventory': " + c.count());
        FindIterable<Document> docs = c.find();
        for (Document doc : docs)
            System.out.println(doc.toJson());
        mongo.close();
    }
}
```

UNIVERSIDADE DE AVEIRO

Java driver – example 1 output

```
--- Colecções na BD 'test'
invoice
inventory
collection
orders
--- Total de documentos em 'countries': 5
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c432c" }, "item" : "journal",
"qty" : 25.0, "status" : "A" }
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c432d" }, "item" : "notebook",
"qty" : 50.0, "status" : "A" }
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c432e" }, "item" : "paper", "qty"
: 100.0, "status" : "D" }
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c432f" }, "item" : "planner",
"qty" : 75.0, "status" : "D" }
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c4330" }, "item" : "postcard",
"qty" : 45.0, "status" : "A" }
```

Java driver – example 2 (insert)

```java
public class Test2 {
    public static void main(String[] args) {
            // remove log in the console
        java.util.logging.Logger.getLogger("org.mongodb.driver").setLevel(
                Level.SEVERE);
        MongoClient mongo = new MongoClient("localhost", 27017);
        MongoCollection<Document> coll =
            mongo.getDatabase("test").getCollection("inventory");

        Document doc = new Document("item", "database")
            .append("qty", 1)
            .append("status","M");
        coll.insertOne(doc);
        FindIterable<Document> docs = coll.find();
        for (Document d : docs)
            System.out.println(d.toJson());
        mongo.close();
    }
}
```

Java driver – example 2 output

```
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c432c" }, "item" : "journal",
"qty" : 25.0, "status" : "A" }
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c432d" }, "item" : "notebook",
"qty" : 50.0, "status" : "A" }
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c432e" }, "item" : "paper", "qty"
: 100.0, "status" : "D" }
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c432f" }, "item" : "planner",
"qty" : 75.0, "status" : "D" }
{ "_id" : { "$oid" : "59b1b730935c2a0ca72c4330" }, "item" : "postcard",
"qty" : 45.0, "status" : "A" }
{ "_id" : { "$oid" : "59b2a7e98cbca6f6497c7110" }, "item" : "database",
"qty" : 1, "status" : "M" }
```

Java driver – example 3 (multi-doc)

```java
public class Test3 {
    public static void main(String[] args) {
        java.util.logging.Logger.getLogger("org.mongodb.driver").setLevel(
               Level.SEVERE);
        MongoClient mongo = new MongoClient("localhost", 27017);
        MongoCollection<Document> coll =
             mongo.getDatabase("test").getCollection("inventory");
        Document doc = new Document("item", "record")
             .append("size",
                new Document("h", 10).append("l", 20).append("w", 30))
             .append("qty", 1)
             .append("status","R");
        coll.insertOne(doc);
        FindIterable<Document> docs = coll.find(new Document("status", "R"));
        for (Document d : docs)
             System.out.println(d.toJson());
        mongo.close();
    }
}
```

```
{ "_id" : { "$oid" : "59b2a9eb8cbca6f6527068b2" }, "item" : "record",
"size" : { "h" : 10, "l" : 20, "w" : 30 }, "qty" : 1, "status" : "R" }
```

Summary

- Document Database
- MongoDB
  - JSON document database
  - Sharding with master-slave replication architecture
- Query functionality
  - CRUD operations
    - Insert, find, update, remove
  - Complex filtering conditions
  - Index structures
  - MapReduce
- Java driver

Resources

- Eric Redmond, Jim R. Wilson. ***Seven databases in seven weeks***, Pragmatic Bookshelf, 2012.
- Martin Kleppmann, ***Designing Data-Intensive Applications***, O'Reilly Media, Inc., 2017.
- Pramod J Sadalage and Martin Fowler, ***NoSQL Distilled*** Addison-Wesley, 2012.

- MongoDB Docs, https://docs.mongodb.com
- Java Driver, http://mongodb.github.io/mongo-java-driver/