



Tom Wiesing

Databases & Web Applications

November 2, 2015


Overview

- ▶ Introduction & Structure of MongoDB
- ▶ CRUD operations & Basic querying
- ▶ Aggregation Operations
- ▶ Advanced Write Operations
- ▶ Summary & Further topics

Introduction (1): Document-oriented databases

- ▶ document-oriented databases
 - ▶ stores documents
 - ▶ create, read, update and delete documents
- ▶ documents do **not** have to be of the same shape
 - ▶ oriented on the data itself, not the shape
 - ▶ flexible w.r.t. data inside them
 - ▶ missing / optional values easy to incorporate
 - ▶ easy to change format of data on-demand
 - ▶ referential integrity (if applicable) difficult

Introduction (2): Structure of MongoDB

- ▶  **mongoDB** is an open-source, document database designed for ease of development and scaling
 - ▶ so it is document-oriented
- ▶ stores documents so-called “records”
 - ▶ documents are essentially JSON, i.e. key-value pairs
 - ▶ keys are strings
 - ▶ values can for example be strings, numbers, other documents, arrays of values, arrays of other documents

Introduction (3): Structure of MongoDB

```
{
  "_id" : ObjectId("54c955492b7c8eb21818bd09"),
  "address" : {
    "street" : "2 Avenue",
    "zipcode" : "10075",
    "building" : "1480",
    "coord" : [ -73.9557413, 40.7720266 ],
  },
  "borough" : "Manhattan",
  "cuisine" : "Italian",
  "grades" : [
    {
      "date" : ISODate("2014-10-01T00:00:00Z"),
      "grade" : "A",
      "score" : 11
    },
    {
      "date" : ISODate("2014-01-16T00:00:00Z"),
      "grade" : "B",
      "score" : 17
    }
  ],
  "name" : "Vella",
  "restaurant_id" : "41704620"
}
```

Introduction (4): Structure of MongoDB

- ▶ each mongodb server can host several databases
 - ▶ like in MySQL
- ▶ each database can host several collections
 - ▶ similar to tables
- ▶ each collection contains a set of documents
 - ▶ **but:** documents do not need to be of the same shape

CRUD operations (1): Via the shell

- ▶ CRUD (Create, Read, Update, Delete) in MongoDB
 - ▶ You need to be able to do something with the database
- ▶ we will use the shell for now
- ▶ Queries themselves are JSON
 - ▶ we use a version of javascript to specify the operation itself
- ▶ enter them in an interactive shell (the “mongo” executable)
 - ▶ has autocompletion

CRUD operations (2): Selecting Database & Collection

- ▶ `show databases`
 - ▶ shows available databases
- ▶ `use some_awesome_database`
 - ▶ switches to a database
- ▶ `show collections`
 - ▶ show all collections in the current database

CRUD operations (3): Find, insert, update, remove

- ▶ `db.my_collection.find(query)`
 - ▶ find everything that matches the query
- ▶ `db.my_collection.insert(documents)`
 - ▶ insert new documents into a collection
- ▶ `db.my_collection.update(query, change)`
 - ▶ update every document that matches the query
- ▶ `db.my_collection.remove(query)`
 - ▶ remove every document that matches the query

Basic queries (1)

- ▶ `db.my_collection.find({field:value})`
 - ▶ match fields exactly (like = in SQL)
- ▶ `db.my_collection.find({field:regex})`
 - ▶ match regular expressions (like LIKE in SQL)
- ▶ `db.my_collection.find({field:{$gt:value}})`
 - ▶ greater than (use \$gte for greater or equal)
- ▶ `db.my_collection.find({field:{$lt:value}})`
 - ▶ less than (use \$lte for greater or equal)
- ▶ `db.my_collection.find({field:{$ne:value}})`
 - ▶ not equal to
- ▶ You can combine queries in the same object
 - ▶ works like a logical and
- ▶ There are also logical operators \$and and \$or
 - ▶ we do not want to go into too much detail here

Time for a short demo

Aggregation Operations (1)

- ▶ Aggregations process data records and return computed results
- ▶ in mongodb there are three types of aggregations
 - ▶ Single Purpose Aggregation Operations
 - ▶ Map-Reduce
 - ▶ Aggregation Pipelines
- ▶ we will only talk about these briefly

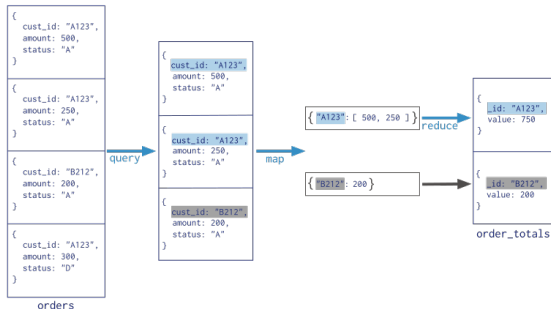
Aggregation Operations (2): Single Purpose Aggregation Operations

- ▶ `db.my_collection.count(query)`
 - ▶ Count the number of documents that match
- ▶ `db.my_collection.distinct(field)`
 - ▶ return an array of the distinct values of the field
- ▶ `db.my_collection.group(query)`
 - ▶ groups documents, supports aggregation-pipeline like operations

Aggregation Operations (3): Map-Reduce

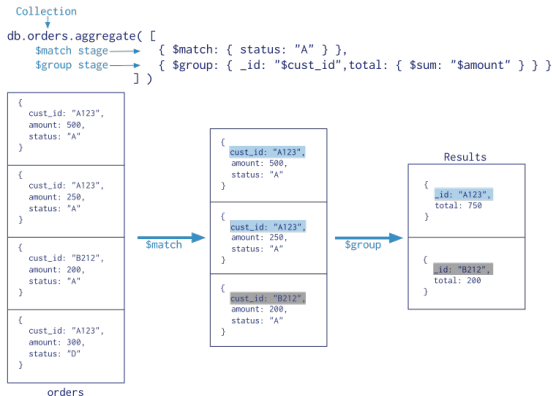
- ▶ is a general data processing paradigm
 - ▶ filter & sort data using Map, then summarise it Reduce

```
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" }
  )
```



Aggregation Operations (5): Aggregation Pipeline

- ▶ modeled on the concept of data processing pipelines
 - ▶ multi-stage pipeline as an alternative to Map-Reduce



Advanced Write Operations (1): Atomicity

- ▶ MongoDB write operations are atomic on the level of a document
 - ▶ if two properties are update don one document nothing can happen in between
- ▶ there are several so called “write concerns”
 - ▶ Guarantee that something is actually written to the database
 - ▶ can provide different levels of guarantee

Advanced Write Operations(2): Write concerns

- ▶ there are four levels
 - ▶ Unacknowledged
 - ▶ returns immediately
 - ▶ Acknowledged (default)
 - ▶ write to the database then return
 - ▶ Journalled
 - ▶ write to Journal (= disk) then return
 - ▶ Replica Acknowledged
 - ▶ write to database and backups then return

Advanced Write Operations (3): Transactions

- ▶ sometimes more than a single operation needs to be atomic
 - ▶ for example transfer money from one account to another
- ▶ we want to ensure that either all or none of the operations are run
 - ▶ we run them in one so-called transaction
- ▶ MongoDB does this with so-called “Bulk Write Operations”
 - ▶ we do not go into details here

Conclusion (1) : Summary

- ▶ MongoDB is a document oriented database
 - ▶ Databases, Collections, Documents
- ▶ Queries can be done using a javascript-style syntax
 - ▶ documents are JSON
- ▶ All queries & write operations are JSON
 - ▶ use javascript only for determining the operation itself
- ▶ Easy to get started, supports also more complicated operations

Conclusion (2) : Further topics

- ▶ Integration into programming languages
 - ▶ perfect for JavaScript environments like node
- ▶ Indexing
 - ▶ be faster when searching
- ▶ Mongoose: Object-oriented mapper for mongo
 - ▶ when you want to have a schema
- ▶ ...

The end

Thank you for your attention!

Any Questions, Comments, etc?

► Sources:

- <https://docs.mongodb.org/manual/>
- <https://en.wikipedia.org/wiki/MongoDB>

► Image Sources:

- <https://www.mongodb.com/brand-resources>
- https://docs.mongodb.org/manual/_images/map-reduce.png
- https://docs.mongodb.org/manual/_images/aggregation-pipeline.png