MongoDB

Thomas Schwarz, SJ

MongoDB History

- 2007 Developed by 10gen as a Platform as a Service (PaaS)
- 2009 Open Source model is adopted
- 2013 10gen becomes MongoDB
- 2019 MongoDB as a service on Alibaba cloud
  - MongoDB comes from humongous

Design

- Document based database
  - Records are stored as documents
    - JSON format
      - Javascript Object format
      - Stored internally in a BSON (binary) format

Design

- JSON: series of structured key-value pairs

  -
    ```
    {
       "name": "Emile",
       "age":  64,
       "address":
            {"street":  "Rue de Grenelles 42",
             "City":    "Paris VI"
             "Country": "France"
            }
       "hobbies":   [
           {"name":     "cooking"},
           {"name":     "reading"},
           {"name":     "chess"}
           ]
    }
    ```

Design

- Documents are rich data structures
  - Fields can be
    - Typed
    - Arrays
    - Arrays of sub-documents

Design

- MongoDB
  - Each installation has one or several databases
  - Each database has one or more collections
  - Each collection has one or more (usually many) JSON document

Design

- Collections have no schema as JSON documents have no schema
- If you come from a relational database world, you need to "denormalize" relations

Example

- Information in the employees database
  - We want to join a lot of tables to have data on employees

```
{ "emp_no"     : 10000,
  "first_name" : "Luigi",
  "last_name"  : "Nguyen",
  "birth_date" : "1971-04-12",
  "gender"     : "M",
  "hire_date   : "1993-01-01",
  "contracts   : [ {from_date : "1993-01-01",
                     to_date   : "1993-12-31" ,
                     department: "Research",
                     salary    : 38095,
                     title     : Engineer 1}    },
                   {from_date : "1994-01-01",
                     to_date   : "1994-12-31" ,
                     department: "Research",
                     salary    : 38125,
                     title     : Engineer 1}    }
                  ]
}
```

Design

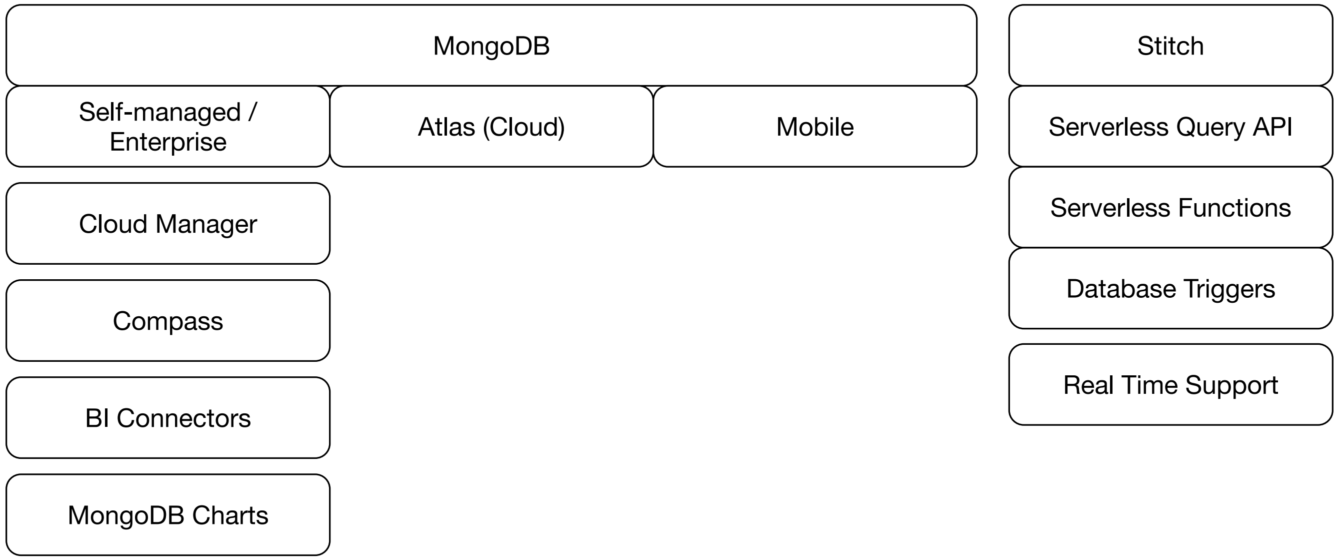
- Advantages of Non-SQL
    - Large Scale: Easier parallelism
        - Often by lowering guarantees: non-transactional
    - Handling of semi-structured data
    - Integration of different databases
    - Either distribution
- Disadvantages
    - Not as universal a tool

Design

- JSON was developed for platform independent data exchange
  - JSON <— JavaScript Object Notation
  - Networks have enough capacity to handle bigger data objects
- MongoDB uses BSON
  - Binary jSON
    - Binary data
    - Extends JSON datatypes
      - e.g. ObjectID('hello world')
    - More efficient storage than just strings

MongoDB Ecosystem

- MongoDB comes in:
  - Self-managed or Enterprise edition
  - Free community version
  - Atlas cloud solution
  - Mobile for simple devices



MongoDB

Self-managed / Enterprise

Atlas (Cloud)

Mobile

Cloud Manager

Compass

BI Connectors

MongoDB Charts

Stitch

Serverless Query API

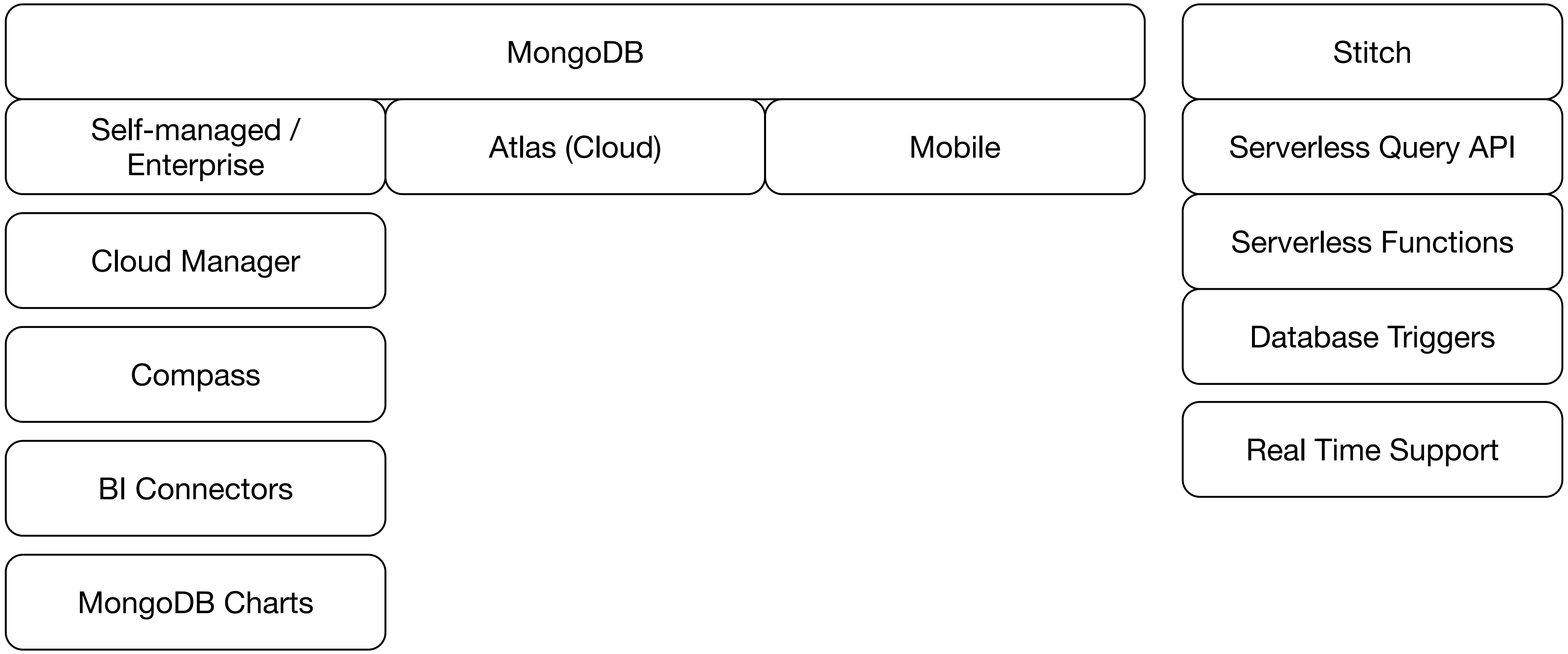
Serverless Functions

Database Triggers

Real Time Support

MongoDB Ecosystem

- Compass: Graphical user interface
- BI connectors and MongoDB charts for data science



MongoDB

Self-managed / Enterprise

Atlas (Cloud)

Mobile

Cloud Manager

Compass

BI Connectors

MongoDB Charts

Stitch

Serverless Query API

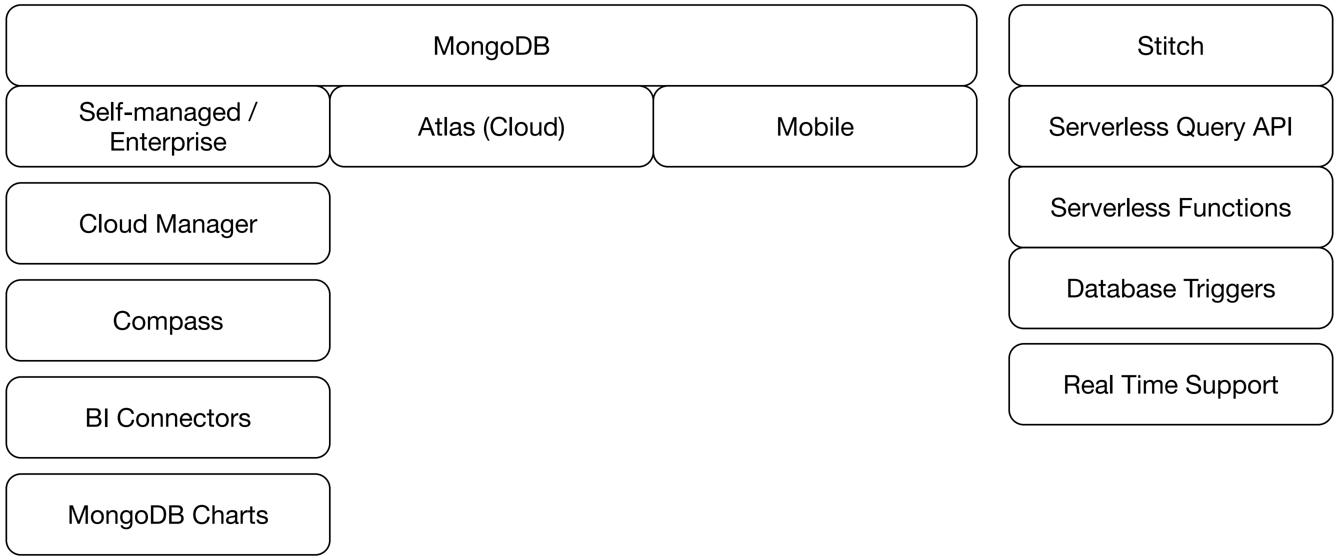
Serverless Functions

Database Triggers

Real Time Support

MongoDB Ecosystem

- Stitch: Server-less back-end solution
  - Includes a serverless query API
  - Serverless functions corresponds to AWS Lambda
  - Database triggers
  - Real time synchronization between database in a cloud and mobile offline databases

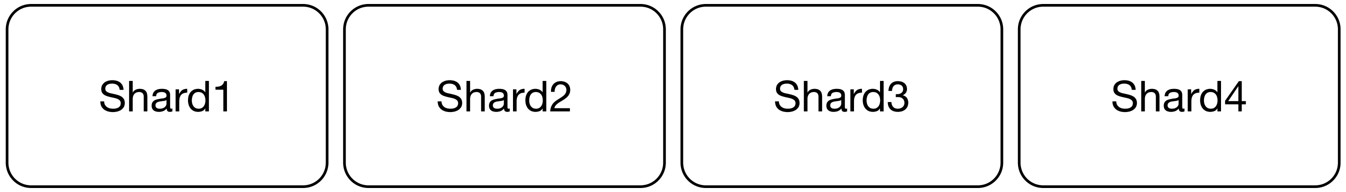


MongoDB Compass

- Download MongoDB compass
- Run a MongoDB instance
- Connect MongoDB compass to the local MongoDB server
- Easier interface than the shell

MongoDB Internals

- Horizontally scalable



Shard1 Shard2 Shard3 Shard4

  - Sharding based on:
    - Hashing
    - Range-based
    - Location-aware
  - Capacity can be adjusted automatically
  - Automatic balancing

MongoDB Internals

- Replication: 2 — 50 copies
    - Primary and secondary copy strategy
        - Updates to primary copy, then broadcast to secondary copies
- Self-healing shards
- Location aware (which data center you are in)

MongoDB Internals

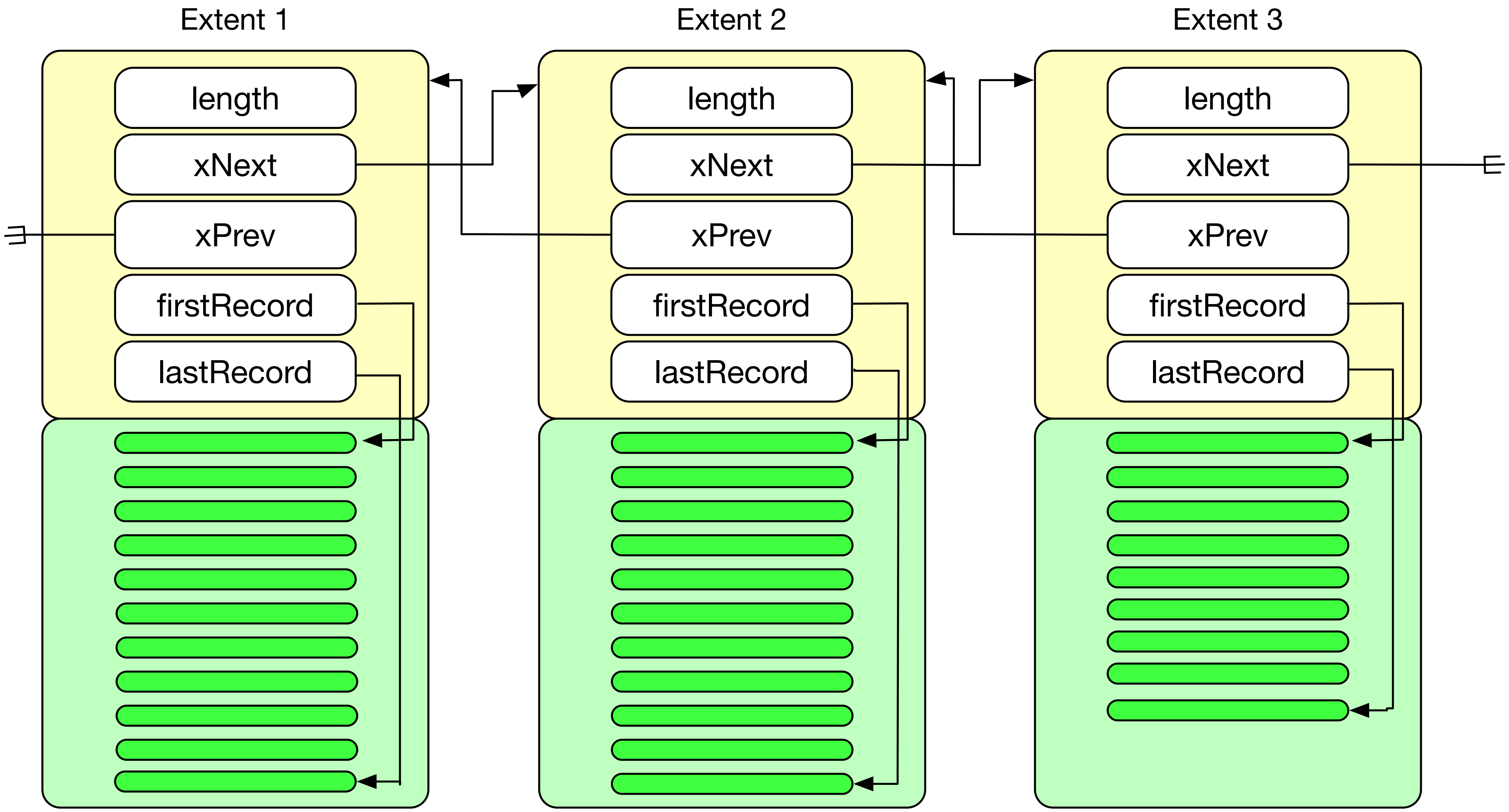
- Storage layer
  - Different workloads require different storage strategies
    - Latency
    - Throughput
    - Concurrency
    - Costs
  - Storage Engine API
    - allows to mix storage engines

MongoDB Internals

- Storage Layer:
    - WT — WiredTiger
        - Up to 80% compression
    - MMAP
        - for read-heavy applications
        - Data is paged into RAM
    - Encrypted Storage Engine
        - End-to-end encryption for sensitive data
    - In memory storage

MongoDB Internals

- MMAP: collections organized into extents



- Extent grows up to 2 GB

MongoDB Internals

- Indices are B-Tree structures
  - Stored in the same files as data but use own extents
  - Look at them using db.stats( )

MongoDB Internals

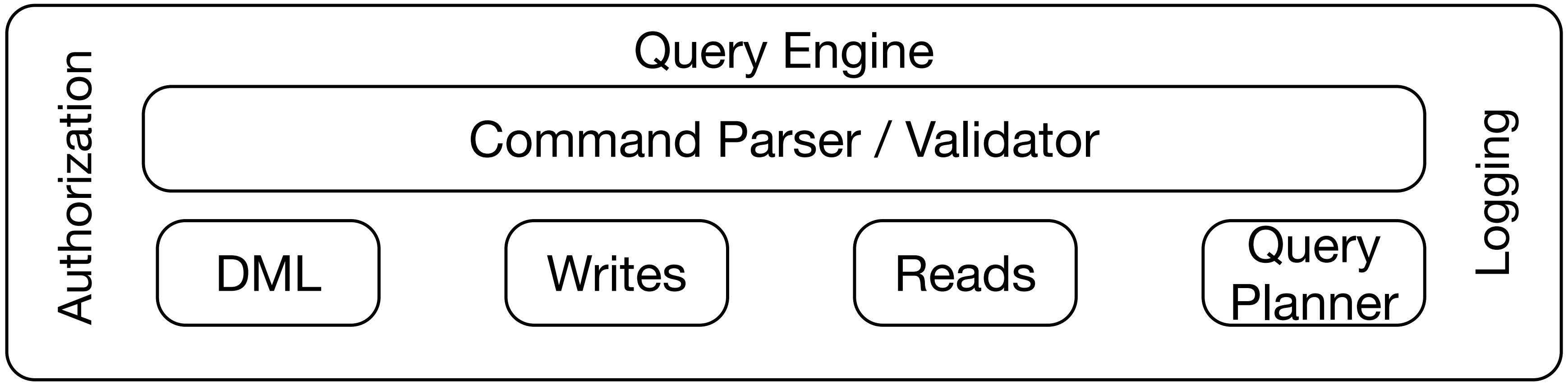
- All data files are memory mapped to Virtual Memory by the OS
- MongoDB just reads and writes to RAM in the file system cache
- OS takes care of the rest
    - Size issue for 32b architectures
    - Corruption solved by journaling (write ahead log)
        - Hard crash can loose a journal flush (100ms)

MongoDB Internals

- Fragmentation
    - If records are deleted holes develop that cannot always be filled

MongoDB Internals

- Query engine



Installing MongoDB

- MongoDB installer at Mongodb.com
  - Windows: download installer and install mongodb as a service
  - MacOS: search from macos mongodb brew installation
    - Need to get homebrew first

Getting started

- Start mongodb:

```
thomasschwarz@Peter-Canisius ~ % mongo
```

- Look at databases

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Getting Started

- Create a database / switch to it

```
> use shop
```

- Create a document

```
> db.products.insertOne({"name": "widget", price: 5.32})
```

- Look at it

```
> db.products.find()
```

Getting Started

- Can use interfaces with many languages
    - Python: Use pip to install pymongo

Getting Started

- Let's work with the shell first:
    - Here were our commands to start out

```
> use shop

> db.products.insertOne({"name": "widget", price: 5.32}

> db.products.find()
```

- If we insert something more, we get

```
db.products.insertOne({name: "A book", price: 9.98})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e8fe8a45b3c2a47a070a1e7")
}
```

- there is an automatic object id that is created

Getting Started

- db.products.find( ) finds all entries in db.products
  - Using db.products.find( ).pretty( ) gives all the objects in a slightly more readable format

```
> db.products.find().pretty()
{
   "_id" : ObjectId("5e6484e6575cfc1a39adfc22"),
   "name" : "widget",
   "price" : 5.32
}
{
   "_id" : ObjectId("5e8fe8a45b3c2a47a070a1e7"),
   "name" : "A book",
   "price" : 9.98
}
```

Getting Started

- The _id field is automatically generated
    - But we could define it ourselves

```
toinsert = { _id: ObjectID("adfwrqeeeqwwewe"),
             name: "James Bond",
             designation: "007",
             licence: "to kill")
```

CRUD Operations

- Create
  - insertOne(data, options)
  - insertMany(data, options)
- Update
  - updateOne(filter, data, options)
  - updateMany(filter, data, options)
- Read
  - find(filter, options)
  - findOne(filter, options)
- Delete
  - deleteOne(filter, options)
  - deleteMany(filter, options)

CRUD Operations

- For these exercises:
  - Create a clean slate by dropping any database that you are working with:
  -
    ```
    > show dbs
    admin   0.000GB
    config  0.000GB
    local   0.000GB
    shop    0.000GB
    > use shop
    switched to db shop
    > db.dropDatabase()
    { "dropped" : "shop", "ok" : 1 }
    ```

CRUD Operations

- We now create a shop document

```
> use shop
switched to db shop
```

- We verify the current database

```
> db.getName()
shop
```

- We create a new collection articles by inserting

```
> db.inventory.insertOne( {name: "Graham Smith Apple",
type: "Apple", category: "Fruit", price: 0.85, measure:
"each"})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5ea20a0b91a8c104f51d62dd")
}
```

CRUD Operations

- We can also use InsertMany

```
>>> db.shop.inventory.insertMany( [
{name: "Red Delicious", type: "Apple", category: "Fruit", price:
0.65, measure: "each"},
{name: "Fuji", type: "Apple", category: "Fruit", price: 0.99,
measure: "each"},
{name: "California Strawberries", type: "Strawberries",
category: "Fruit", price: 1.59, measure: "bowl"} ] )
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5ea20ea491a8c104f51d62df"),
    ObjectId("5ea20ea491a8c104f51d62e0"),
    ObjectId("5ea20ea491a8c104f51d62e1")
  ]
}
```

CRUD Operations

- We can verify the state of the database:

```
> db.shop.inventory.find()
{ "_id" : ObjectId("5ea20caf91a8c104f51d62de"), "name" :
"Graham Smith Apple", "type" : "Apple", "category" :
"Fruit", "price" : 0.85, "measure" : "each" }
{ "_id" : ObjectId("5ea20ea491a8c104f51d62df"), "name" :
"Red Delicious", "type" : "Apple", "category" : "Fruit",
"price" : 0.65, "measure" : "each" }
{ "_id" : ObjectId("5ea20ea491a8c104f51d62e0"), "name" :
"Fuji", "type" : "Apple", "category" : "Fruit", "price" :
0.99, "measure" : "each" }
{ "_id" : ObjectId("5ea20ea491a8c104f51d62e1"), "name" :
"California Strawberries", "type" : "Strawberries",
"category" : "Fruit", "price" : 1.59, "measure" : "bowl" }
>
```

CRUD Operations

```
> db.shop.inventory.find().pretty()
{
   "_id" : ObjectId("5ea20caf91a8c104f51d62de"),
   "name" : "Graham Smith Apple",
   "type" : "Apple",
   "category" : "Fruit",
   "price" : 0.85,
   "measure" : "each"
}
{
   "_id" : ObjectId("5ea20ea491a8c104f51d62df"),
   "name" : "Red Delicious",
   "type" : "Apple",
   "category" : "Fruit",
   "price" : 0.65,
   "measure" : "each"
}
{
   "_id" : ObjectId("5ea20ea491a8c104f51d62e0"),
   "name" : "Fuji",
   "type" : "Apple",
   "category" : "Fruit",
   "price" : 0.99,
   "measure" : "each"
}
{
   "_id" : ObjectId("5ea20ea491a8c104f51d62e1"),
   "name" : "California Strawberries",
   "type" : "Strawberries",
   "category" : "Fruit",
   "price" : 1.59,
   "measure" : "bowl"
}
```

CRUD Operations

- Inserts:
  - insertOne( ) inserts a single document
    - `db.persons.insertOne({name: "Emil", age: 64})`
  - insertMany with an array of documents
    - `db.persons.insertMany([{name: "Mary", age:50}, {name: "Fred", age: 58, hobbies: ["hiking", "drinking"]}])`
  - insert( ) does the same as insert or insertMany, but does not return a result in the shell
  - mongoimport imports a json array from the file system

CRUD Operations

- Insert operations either generate their own IDs or you provide them
    - `db.persons.insertOne({_id: 12345, name: "Emil", age: 64})`
        - Notice the underscore before id
    - Checks whether the user-provided ID is unique

CRUD Operations

- Ordered Inserts
  - If there is an error on multiple inserts
  - Stop the current insert opertion
  - Does not roll-back previous inserts
- To override the behavior, set options for insert

  -
    ```
    db.person.insertMany([{_id: 12345, name:
    "bubu", age: 5}, {_id: 12346, name: "Yogi",
    age: 6}, {ordered: false}])
    ```

CRUD Operations

- Find
  - db.collection.find({key: value})
  -
    ```
    > db.zip.find({"city": "MILWAUKEE"})
    { "_id" : "53202", "city" : "MILWAUKEE", "loc" : [ -87.896792,
    43.050601 ], "pop" : 20178, "state" : "WI" }
    { "_id" : "53203", "city" : "MILWAUKEE", "loc" : [ -87.915375,
    43.040299 ], "pop" : 456, "state" : "WI" }
    { "_id" : "53204", "city" : "MILWAUKEE", "loc" : [ -87.931685,
    43.015778 ], "pop" : 41978, "state" : "WI" }
    { "_id" : "53221", "city" : "MILWAUKEE", "loc" : [ -87.944734,
    42.954864 ], "pop" : 35767, "state" : "WI" }
    { "_id" : "53223", "city" : "MILWAUKEE", "loc" : [ -87.989818,
    43.162374 ], "pop" : 30272, "state" : "WI" }
    ```

CRUD Operations

- Can use comparison operators
  - https://docs.mongodb.com/manual/reference/operator/query-comparison/
  - $eq, $gt, $gte, $in, $lt, $lte, $ne, $nin

```
db.zip.find({"pop": {$lt: 100}})
```

CRUD Operations

- Find can also be used to look for fields in embedded documents
    - E.g. if rating is the name of a subdocument with a key average, you can use
        -
          ```
          db.movies.find({ "rating.average": {$lt: 5}})
          ```

CRUD Operations

- Other find features:
  - Logical connectors
  - Array querying
  - Regular expression
  - Evaluation of a boolean expression ($expr)

CRUD Operations

- Results of find are given by a "cursor"
    - Cursor results can be counted, printed, …, or sorted
- Cursors are "manually" handled in a programming environment (pymongo)

CRUD Operations

- Updates
  - Use updateOne, updateMany
  - First part is a filter
  - Second part is an update operation

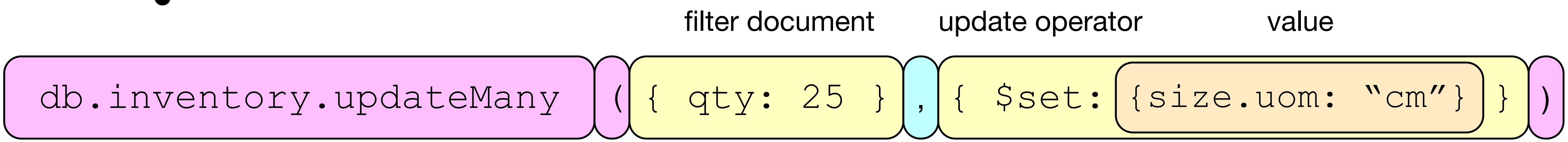
CRUD Operations

- Example (from manual)

```
db.inventory.insertMany( [
   { item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" },
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },
   { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },
   { 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" },
   { item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" }, status: "A" }
] );
```

CRUD Operations

- updateOne updates the first document that fits the filter condition
- updateMany updates all documents that fit the filter condition
- replaceOne replaces a document that fits the filter
-



filter document

update operator

value

```
db.inventory.updateMany ( { qty: 25 } , { $set: {size.uom: “cm”} } )
```

CRUD Operations

- Other update operators:
  - $inc increments a field
  - $currentDate sets a field to the current time
  - $min only updates if the specified value is less than the existing value
  - $max
  - $mul multiplies the value of a field
  - $unset: removes a specified field
  - …

CRUD Operations

- Delete
    - deleteOne, deleteMany
    - Filter document determines the selection

Schemas and Relations

- MongoDB allows us to :
    - Structure all our documents in the same manner
        - Almost like a RDBMS table
    - Structure all our documents in completely different manners

Schemas and Relations

- Schemas
    - MongoDB allows the use of validators
        - E.g. javascripts that check the structure of a document to be inserted
        - Administrator can enable validation
            - With different extent (updates / inserts) and actions (default is error, warning)
        - Documents that violate the validator are not inserted/updated

`https://docs.mongodb.com/manual/core/schema-validation/`

Schemas and Relations

- Data Modelling:
  - Organize data for operations
    - data fetch
    - data writes
  - Organize data for size

Schemas and Relations

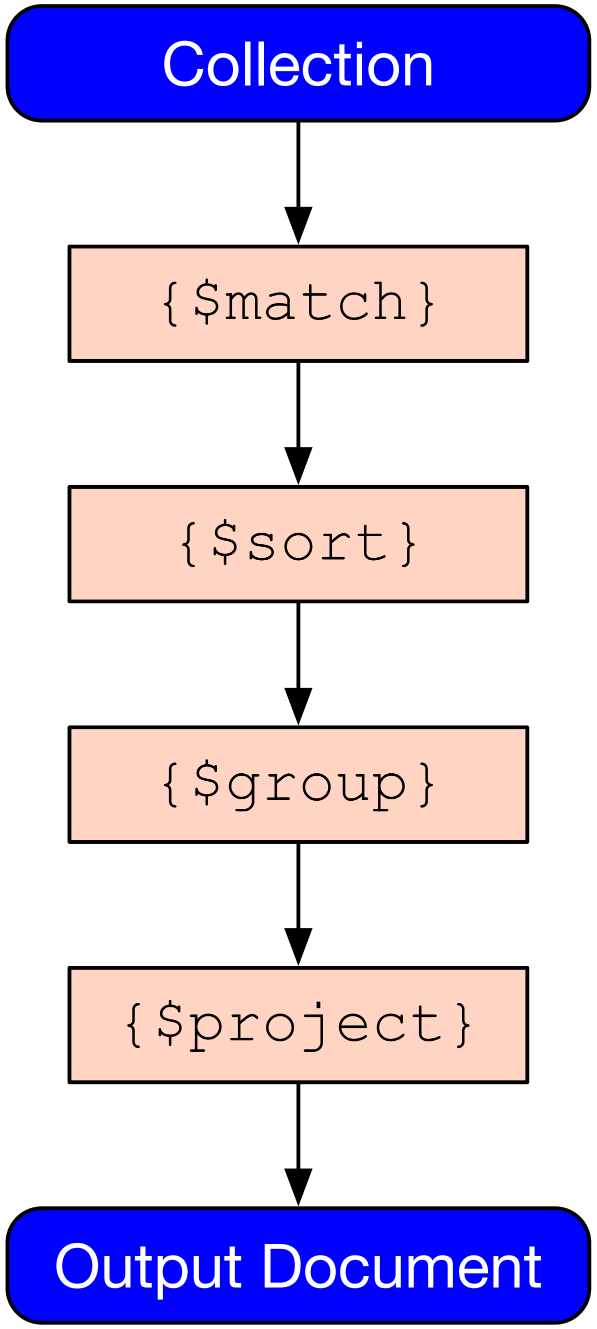
- Embedding documents
  - MongoDB allows embedding of documents
    - E.g.: Order can include the product description
    - Up to generous limits on document size and embedding levels
  - MongoDB allows references to documents
    - E.g.: Order can include the reference to the product description

Schemas and Relations

- Organize data for operations:
  - Fetches dominate
    - Try to keep all data together
    - Duplicate
    - Embed documents
      - Even though this leads to update anomalies
  - Writes dominate
    - Avoid duplication
    - Do not embed documents
      - Especially if they might change

Aggregation

- Aggregation Framework
  - Various stages applied on a collection
  - Stages can be repeated
  - `db.collection.aggregate([{stage1}, {stage2}, …])`



Aggregation

- $lookup: Stage that allows combining two collections
  - Slow, but powerful

Aggregation

- Example:

```
customers.aggregate([
  {  $lookup: {
        from: "Address",
        localField: "address",
        foreignField: "_id"
        as: "addressData"
        }
   }
])
```

- Creates a list of clients with embedded addresses

Clients

```
{
   userName: "Thomas",
   address: id1
}
```

Address

```
{
   _id: "id1"
   city: "Milwaukee"
   street: "1345 W Wells St"
   zip: 54323
}
```

Aggregation

- from : The collection that you are joining with
- localField: the name of the joining attribute in the local collection
- foreignField: the name of the joining attribute in the other (from) collection
- as: name of the key

Transactions

- Mongo 4.0 allows transactions
  - Need to have sessions and replicas
  - Can commit in a session

Geospatial Queries

- MongoDB can deal with geospatial data effectively
  - Stores in GeoJSON format
    - Example: Golden Gate Park
    - type has to be "Point"
    - coordinates are longitude, latitude (in this order)

```
{type: "Point", coordinates: [-122.445, 37.767]}
```

Geospatial Queries

For $near to work, we need an index

```
db.places.createIndex({location: "2dsphere"})
```

Now we can use it to find near places with 1000 meters

```
db.places.find({loc: {$near: {$geometry:
{type: "Point", coordinates: [-122,45, 37.77]}},
$maxDistance: 1000}})
```

Exercise

- Import the zipcodes database from
  - http://media.mongodb.org/zips.json
- Store it in a known directory, e.g. Downloads
  - You can check what it looks like:

```
{ "_id" : "53222", "city" : "MILWAUKEE", "loc" : [ -88.02687, 43.08283 ],
"pop" : 25406, "state" : "WI" }
{ "_id" : "53223", "city" : "MILWAUKEE", "loc" : [ -87.989818, 43.162374 ],
"pop" : 30272, "state" : "WI" }
{ "_id" : "53224", "city" : "MILWAUKEE", "loc" :
[ -88.03274399999999, 43.159415 ], "pop" : 18182, "state" : "WI" }
{ "_id" : "53225", "city" : "MILWAUKEE", "loc" : [ -88.03464, 43.115416 ],
"pop" : 25395, "state" : "WI" }
```

Exercise

- To make this into a MongoDB database, you need to use **<u>a different terminal window</u>**
- Use mongoimport

```
% mongoimport --db=zipcodes --collection=zip --file="zips.json"
2020-04-24T15:39:57.253-0500connected to: mongodb://localhost/
2020-04-24T15:39:57.588-050029353 document(s) imported successfully.
0 document(s) failed to import.
```

  - you generate a new database: zipcodes
  - you generate a new collection in the database : zip

Exercise

- Now check that the import worked

```
> show dbs
admin     0.000GB
config    0.000GB
local     0.000GB
shop      0.000GB
zipcodes  0.002GB
> use zipcodes
switched to db zipcodes
> show collections
zip
```

Exercise

- Find zip codes with a population of less than 500

Exercise

```
> db.zip.find({"pop": {$lt: 100}})
{ "_id" : "01338", "city" : "BUCKLAND", "loc" : [ -72.764124, 42.615174 ],
"pop" : 16, "state" : "MA" }
{ "_id" : "01350", "city" : "MONROE", "loc" : [ -72.960156, 42.723885 ],
"pop" : 97, "state" : "MA" }
{ "_id" : "02163", "city" : "CAMBRIDGE", "loc" : [ -71.141879, 42.364005 ],
"pop" : 0, "state" : "MA" }
{ "_id" : "02713", "city" : "CUTTYHUNK", "loc" : [ -70.87854, 41.443601 ],
"pop" : 98, "state" : "MA" }
{ "_id" : "02815", "city" : "CLAYVILLE", "loc" : [ -71.670589, 41.777762 ],
"pop" : 45, "state" : "RI" }
```