CSCU9YQ - NoSQL Databases
Lecture 2: Intro to MongoDB

Gabriela Ochoa

Introduction

- Cross-platform, open-source document database
- Provides
  - high performance,
  - high availability,
  - automatic scaling
- Useful links
  - https://www.mongodb.com/
  - https://docs.mongodb.com/manual/
  - https://university.mongodb.com/ (free learning material)

MongoDB Strengths

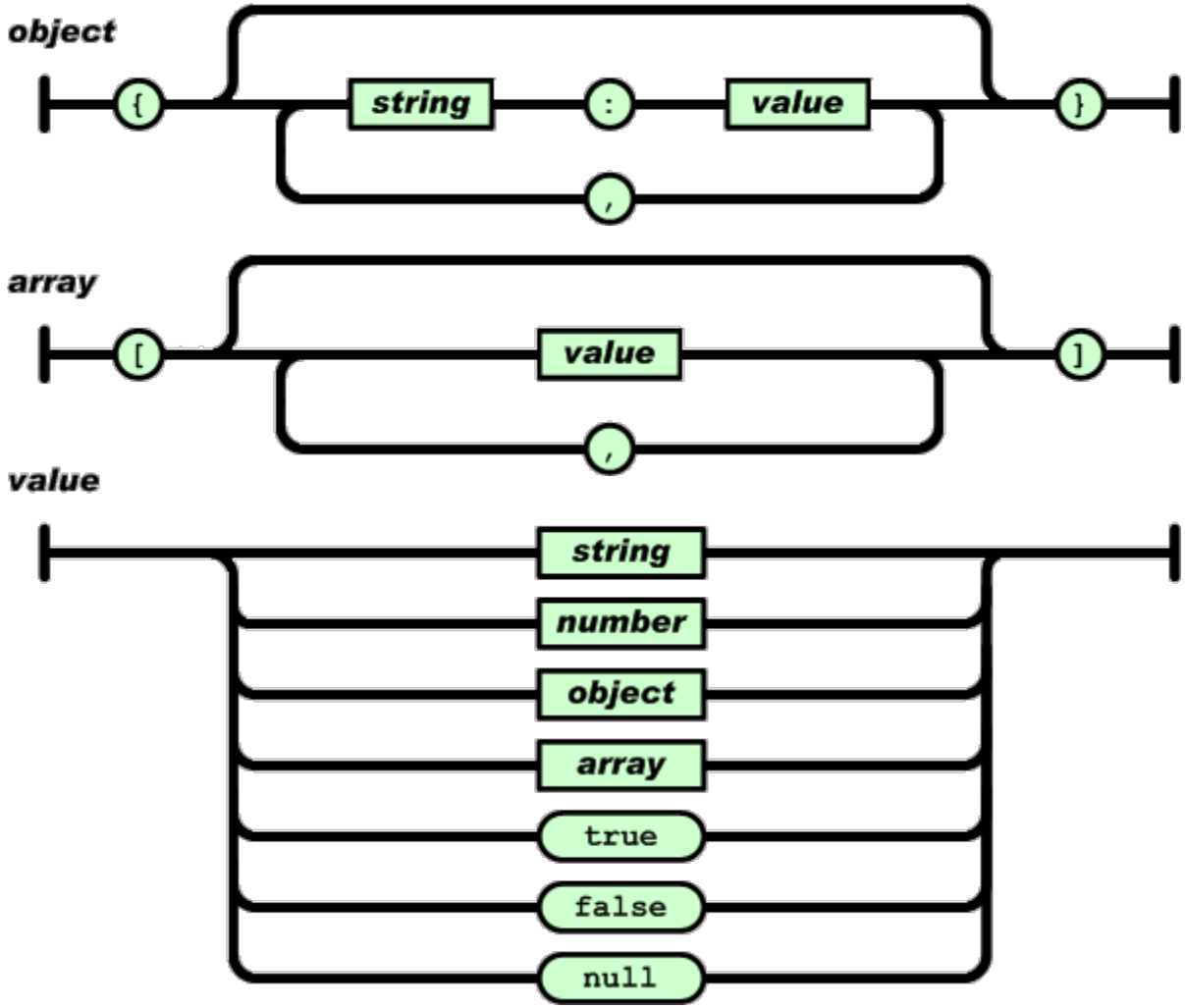
- Stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time
- The document model maps to the objects in your application code, making data easy to work with
- Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyse your data
- Distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use

Document Database

```
{
  name: "sue",                          ← field: value
  age: 26,                              ← field: value
  status: "A",                          ← field: value
  groups: [ "news", "sports" ]          ← field: value
}
```

- A record in MongoDB is a document, similar to JSON objects
- Data structure composed of field and value pairs
- The values of fields may include other documents, arrays and arrays of documents

Summary of JSON



BSON (short for Binary JSON)

- MongoDB stores data records as BSON doccs.
- BSON is a binary representation of JSON docs, though it contains more data types than JSON
- For example, BSON has a Date type and a BinData type.
- Like JSON, BSON supports the embedding of docs and arrays within other docs and arrays
- Useful links
  - http://bsonspec.org/
  - https://docs.mongodb.com/manual/reference/bson-types/

Document Structure

- MongoDB documents are composed of field-and-value pairs

```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

Example of MongoDB Document

```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
               death: new Date('Jun 07, 1954'),
               contribs: [ "Turing machine", "Turing test” ],
               views : NumberLong(1250000)
            }
```

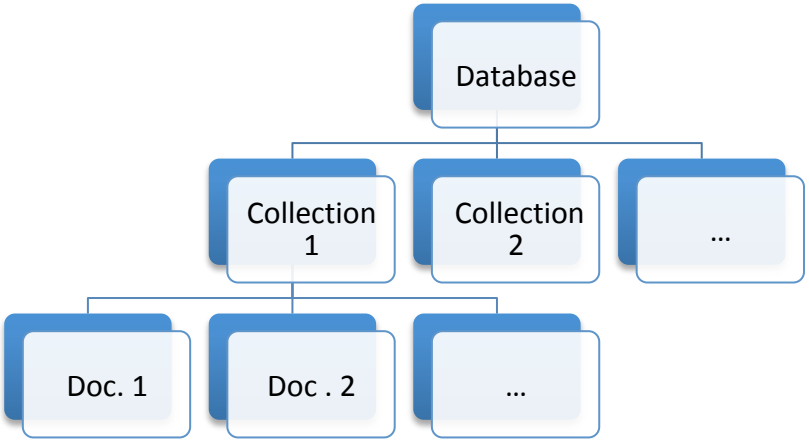
- `_id` holds an ObjectId.
- `name` holds an embedded document that contains the fields `first` and `last`.
- `birth` and death hold values of the Date type.
- `contribs` holds an array of strings.
- `views` holds a value of the NumberLong type

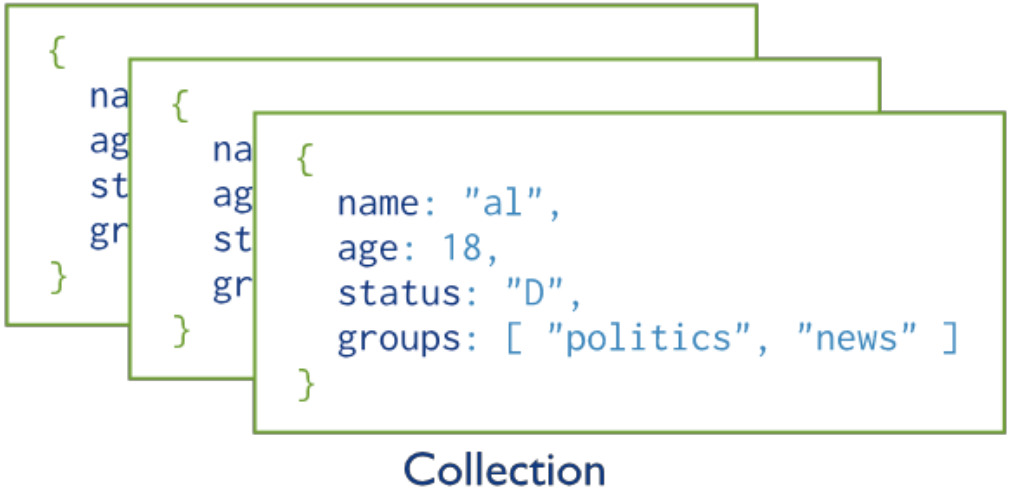
Primary Key: _id field

- Each document requires a unique _id field that acts as a primary key.
- If an inserted document omits the _id field, MongoDB automatically generates an ObjectId
  - 12 bytes, where the first 4 bytes are a timestamp of the ObjectId's creation. Specifically:
    - a 4-byte value representing the seconds since the Unix epoch,
    - a 5-byte random value, and
    - a 3-byte counter, starting with a random value.

Databases and Collections

- BSON documents (data records) are stored in collections, which are analogous to Tables in RD
- Collections are stored in in databases.





Mongo Shell

- Interactive JavaScript shell interface to MongoDB
- Allows us to manipulate databases, collections and documents
- Powerful interface to test queries and operations
- Provides a fully functional JavaScript environment
- User account on the MongoDB server at the University. Run the mongo shell from a DOS command line
  - `mongo –host mqr0.cs.stir.ac.uk –u <username> -p <password>`
- Useful links:
  - https://docs.mongodb.com/manual/reference/mongo-shell/
  - https://mws.mongodb.com/?version=3.6

Query Language

- Rich query language to support CRUD operations
  - Create (Insert) Operations
  - Read Operations
  - Update Operations
  - Delete Operations
  - Bulk Write
- Dot notation is used to access the fields of documents, embedded documents and elements of arrays
- Data aggregation
- Text search and geospatial queries

Create Databases

- To select an existing database to use, in the mongo shell: `use myDB`
- If a database does not exist, MongoDB creates the database when you first store data for that database.
- So, you can switch to a non-existent database and perform the following operation in the mongo shell:

```
use myNewDB

db.myNewCollection1.insertOne( { x: 1 } )
```

The `insertOne()` operation inserts a document and creates both the database myNewDB and the collection myNewCollection1 if they do not already exist.

Create Collections

- MongoDB stores documents in collections. Collections are analogous to tables in RD.
- The `db.createCollection()` method explicitly creates a collection with various options (max size, validation rules).
- If a collection does not exist, MongoDB creates the collection when you first store data for that collection.
- Operations for inserting documents (next slide) create their respective collection if they do not already exist.

```
db.myNewCollection2.insertOne( { x: 1 } )
db.myNewCollection3.createIndex( { y: 1 } )
```

Insert Operations

- Add new documents to a collection.
- Insert operations target a single collection.
- Write operations are atomic on the level of a single document.
- Two operations
  - `db.collection.insertOne()`
  - `db.collection.insertMany()`

```
db.users.insertOne(          ← collection
   {
      name: "sue",           ← field: value
      age: 26,               ← field: value   } document
      status: "pending"      ← field: value
   }
)
```

Example Insert Operations

```
db.inventory.insertOne(
  { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
)
```

- Inserts a new document into the inventory collection.
- If the document does not specify an _id field, MongoDB adds the _id field with an ObjectId value to the new document.

```
db.inventory.insertMany([
  { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
  { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
  { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
])
```

- Inserts 3 new documents into the inventory collection.
- If the document does not specify an _id field, MongoDB adds the _id field with an ObjectId value to each document.

Read Operations

- Retrieve documents from a collection; i.e. queries a collection for documents.
- Method: `db.collection.find()`
- You can specify query filters or criteria that identify the documents to return.
- The shell returns a cursor with the data retrieved, by default the cursor iterates over the first 20.

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```

Examples of Queries

```
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" }
]);
```

- Select all documents in the collection, pass empty doc {} as the parameter
  - `db.inventory.find( {} )`
- Specify equality conditions, use <field>:<value> expressions
  - `db.inventory.find( { status: "D" } )`
- Specify conditions using query operators
  - `db.inventory.find( { status: { $in: [ "A", "D" ] } } )`
- Specify And Conditions
  - `db.inventory.find( { status: "A", qty: { $lt: 30 } } )`

There is more to Query! See examples at:
https://docs.mongodb.com/manual/tutorial/query-documents/

Update Operators

- Modify existing documents in a collection
- Methods
  - `db.collection.updateOne(<filter>, <update>, <options>)`
  - `db.collection.updateMany(<filter>, <update>, <options>)`
  - `db.collection.replaceOne(<filter>, <update>, <options>)`
- Target a single collection
- Write operators are atomic on the level of a single document
- More: https://docs.mongodb.com/manual/tutorial/update-documents/

```
db.users.updateMany(                          ← collection
   { age: { $lt: 18 } },                      ← update filter
   { $set: { status: "reject" } }             ← update action
)
```

Delete Operators

- Modify existing documents in a collection
- Methods
  - `db.collection.deleteMany()`
  - `db.collection.deleteOne()`
- Target a single collection
- Write operators are atomic on the level of a single document
- More: https://docs.mongodb.com/manual/tutorial/remove-documents/

```
db.users.deleteMany(          ← collection
   { status: "reject" }       ← delete filter
)
```

Summary

- Introduced MongoDB
- Structure of Documents (JSON, BSON)
- Databases and collections
- Mongo Shell
- Query Language & CRUD operations