CS144: MongoDB

- NoSQL database
  - Document-oriented database
    * Document ~ JSON object
  - Schema-less: no predefined schema
    * MongoDB will store anything anywhere with no complaint!
    * No normalization or joins
    * Both blessing and a curse
      - Mongoose for ensuring structure in the data
  - No support for transaction
    * Every operation is independent of others
- Document
  - Nested key-value pairs in a JSON(-like) format (~ row in relational database)
    * Supports more data types than JSON
  - Stored as BSON (Binary representation of JSON)
  - The field name \_id is reserved for use as a primary key; its value must be unique in the collection, and may be of any type other than an array
  - If a document does not have \_id when inserted, \_id is automatically added with a unique ObjectId
  - Example

```json
{
  "_id": ObjectId("8df38ad8902c"),
  "title": "MongoDB",
  "description": "MongoDB is NoSQL database",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": 100,
  "comments": [  
    { "user":"lover", "comment": "Great book!" },
    { "user":"hater", "comment": "Worst ever!" }
  ]
}
```

* JSON standard requires double quotes for field names, but it is not enforced by MongoDB
- Collection
-- A group of documents (~ table in relational database)

- Document vs Relation
  - Relational model “flattens” data
    * Set of independent tables
    * Removes redundancy
    * Table is designed based on the intrinsic nature of the data, not for a particular application
    * Efficient join algorithms to synthesize an output desired by the user
  - Document model preserves the view of a particular application
    * Hierarchically nested objects
    * Potential redundancy
    * No need to “decompose” data for storage and “join” them back for retrieval
    * Retrieving data with different “view” is difficult

- Basic MongoDB commands
  - Basic CRUD operations: `insert()`, `find()`, `update()`, and `remove()`
  - `mongo`: start MongoDB shell
  - `show dbs`: show list of databases
  - `use <dbName>`: use the database. create a new one if not exists
  - `db.dropDatabase()`: delete current database
  - `show collections`: show collections
  - `db.createCollection("books")`: create `books` collection
  - `db.collName.drop()`: drop `collName` collection
  - `db.books.insert({title: "MongoDB", likes: 100})`: insert a doc
  - `db.books.find({likes: 100})`: find matching documents
    * `{likes: {$lt: 10}}` (likes < 10), `{likes: {$ne: 100}}` (likes <> 100)
    * `{and: [{likes:{gte: 10}}, {likes:{lte: 20}}]}` (10 <= likes <= 20)
    * first parameter is the condition, second parameter is the update operation
      - `{unset: {likes: ""}}` remove the `likes` field from the matching document
      - `{inc: {likes: 1}}`: increases `likes` field by 1
  - `update values of one matching document`
    - add third parameter `{multi:true}` to update all matching documents
  - `db.books.remove({title: "MongoDB"})`
* remove all matching documents
* add second parameter `{justOne: true}` to remove only one matching document

```javascript
- db.books.createIndex({title:1, likes:-1})
```
* create one index on combined attributes “title” and “likes”
* 1 means ascending order, -1 means descending order