**CSCI 4333.2**

11/18/2024

**Introduction to MongoDB**

by K. Yue

**1. Introduction**

* NoSQL document model distributed database owned by MongoDB (NASDAQ: MDB).
* Documents are stored in JSON format.
* Three versions:
	+ Community server: open source and version
	+ Enterprise server: commercial version
	+ Atlas: cloud version

**1.1 Installation**

For this class, install the followings.

1. MongoDB community server: ensure that it includes Mongo Compass, a MongoDB client, <https://www.mongodb.com/try/download/community>
2. Mongo Shell:
	1. mongosh.exe: a Javascript shell for interacting with MongoDB, <https://www.mongodb.com/try/download/shell>.
	2. Do *not* use mongo.exe, the deprecated former shell.
3. Mongo Compass includes a Mongosh.
4. MongoDB tools: command line utilities including import and export, <https://www.mongodb.com/try/download/database-tools>.
	1. After unzipping, you may put mongosh and these utilities in the same location of the other mongoDB programs, e.g., C:\Program Files\MongoDB\Server\5.0\bin.
	2. You may add the directory “C:\Program Files\MongoDB\Server\5.0\bin”, or similar, in the system PATH variable so these tools can be used anywhere.
5. To be able to use MongoDB through Python, you will to install a driver: "pip install pymongo" in cmd.

**1.2 Server-Client DBMS architecture**

* Like many DBMS, MongoDB uses a client server model.
* Server:
	+ In case the MongoDB server has not been started, run "mongod" in a command terminal.
	+ To check whether mongod is running, execute 'tasklist /FI "IMAGENAME eq mongod.exe"' in Command CLI.
	+ It listens to a port to accept and interpret commands and return results.
	+ mongod's default port: 27017.
* Clients: send MongoDB commands and accept results. Clients used in this course:
	+ Mongo Compass
* 
	+ mongosh
	+ Python through pymongo (if Python is used.)

**1.3 Resources**

* MongoDB manual: <https://docs.mongodb.com/manual/>

**2. MongoDB Structures**

* MongoDB is structured as db -> collection -> document (object, JSON, nested structures) in a way similar to db -> table (flat structure) -> row in relational DB.
* Thus, documents are inserted into a collection of a db.
* db and collection do not need to exist before referencing them.
* In MongoDB's db, within mongosh:
	+ 'use tinker' set the default db to tinker.
	+ The keyword db refers to the default db.
	+ If 'tinker' does not exist, it will be created.

**2.1 Using mongo command CLI through mongosh**

* Run 'mongosh' in command CLI in your working directory.
* Mongosh accept JavaScript commands in a mongo shell setting.
* For inserting documents, it supports two methods, insertOne and insertMany.
* See mongosh CRUD:<https://docs.mongodb.com/mongodb-shell/crud/insert/>.

**3. Writing to Mongo**

1. See CRUD operation in Mongo Guide to begin with: <https://docs.mongodb.com/guides/>.
	1. However, the guide uses the deprecated shell "mongo" instead of "mongosh".
	2. Since mongosh should be used, be mindful of discrepancies.

***Example:***

In mongosh, execute the code:

use tinker
db.test1.insertOne(
   {
      "StudentId" :1,
      "StudentName" : "Joseph Connor"
   }
)

gives the following result:

test> use tinker
switched to db tinker
tinker> db.test1.insertOne (
...     {
.....           "StudentId" :1,
.....           "StudentName" : "Joseph Connor"
.....   }
... )
{
  acknowledged: true,
  insertedId: ObjectId("61e0d5f36753d9628bb4bfa1")
}
tinker> db.test1
tinker.test1

Note:

1. In "db.test1.insertOne (", the '(' must not be put into the next line.
2. If not, mongosh thinks that the current JavaScript statement has ended and you may get:

tinker> db.test1.insertOne
[Function: insertOne] AsyncFunction {
  apiVersions: [ 1, Infinity ],
  serverVersions: [ '3.2.0', '999.999.999' ],
  returnsPromise: true,
  topologies: [ 'ReplSet', 'Sharded', 'LoadBalanced', 'Standalone' ],
  returnType: { type: 'unknown', attributes: {} },
  deprecated: false,
  platforms: [ 0, 1, 2 ],
  isDirectShellCommand: false,
  acceptsRawInput: false,
  shellCommandCompleter: undefined,
  help: [Function (anonymous)] Help
}
tinker> (
...     {
.....           "StudentId" :1,
.....           "StudentName" : "Joseph Connor"
.....   }
... )
{ StudentId: 1, StudentName: 'Joseph Connor' }

In Windows, you may start Compass through the startup manual:



In Mongo Compass (you may enter nothing in the 'Paste your connection string' connect box):



* Note that a field \_id with a system generated object id is created. It is unique and can be served as an id.

If the code is executed one more time, Mongo Compass has:



Note:

1. There are now two Joseph Connor.
2. StuId is not a 'primary key'.
3. Document model is not set-theoretic. Relation model is set-theoretic.

To insert a document 'doc' only when it does not already exist, use something like:

if (db.test1.find(doc).count() == 0) { db.test1.insertOne(doc) }

Note:

1. 'db.test1.find(doc)' finds the documents doc (one document in the example below). It returns a cursor, which is an iterator of the query result.
2. cursor has a method count() to count the result.

The following session illustrates this concept.

Code:

show dbs
db.dropDatabase()
show dbs

// remove tinker
use tinker
db.test1.find()
doc = {
      "StudentId" :1,
      "StudentName" : "Joseph Connor"
}
doc
if (db.test1.find(doc).count() == 0) { db.test1.insertOne(doc) }
db.test1.find()
if (db.test1.find(doc).count() == 0) { db.test1.insertOne(doc) }
db.test1.find()

Session:

tinker> db.test1.find()

tinker> doc = {
...             "StudentId" :1,
...             "StudentName" : "Joseph Connor"
... }
{ StudentId: 1, StudentName: 'Joseph Connor' }
tinker> doc
{ StudentId: 1, StudentName: 'Joseph Connor' }
tinker> if (db.test1.find(doc).count() == 0) { db.test1.insertOne(doc) }
{
  acknowledged: true,
  insertedId: ObjectId("61e0e49e6753d9628bb4bfa5")
}
tinker> db.test1.find()
[
  {
    \_id: ObjectId("61e0e49e6753d9628bb4bfa5"),
    StudentId: 1,
    StudentName: 'Joseph Connor'
  }
]
tinker> if (db.test1.find(doc).count() == 0) { db.test1.insertOne(doc) }

tinker> db.test1.find()
[
  {
    \_id: ObjectId("61e0e49e6753d9628bb4bfa5"),
    StudentId: 1,
    StudentName: 'Joseph Connor'
  }
]

**3.2 Unique Index**

* A unique index can be used to ensure that all documents within the collection must have unique values on the fields.
* This can be used for use cases of inserting the document only if the unique index has an unique value.
* Thus, a unique index can serve as a candidate key (if *it is not missing*) for identifying document in the collection.

***Example:***

Code:

// remove tinker
show dbs
db.dropDatabase()
show dbs
// create index
db.test1.createIndex( { "StudentId": 1 }, { unique: true } )
doc = {
      "StudentId" :1,
      "StudentName" : "Joseph Connor"
}
doc
db.test1.insertOne(doc)
db.test1.insertOne(doc)

Session:

tinker> // create index

tinker> db.test1.createIndex( { "StudentId": 1 }, { unique: true } )
StudentId\_1
tinker> doc = {
...       "StudentId" :1,
...       "StudentName" : "Joseph Connor"
... }
{ StudentId: 1, StudentName: 'Joseph Connor' }
tinker> doc
{ StudentId: 1, StudentName: 'Joseph Connor' }
tinker> db.test1.insertOne(doc)
{
  acknowledged: true,
  insertedId: ObjectId("6570fb99629ad72db73f7bcf")
}
tinker> db.test1.insertOne(doc)
MongoServerError: E11000 duplicate key error collection: tinker.test1 index: StudentId\_1 dup key: { StudentId: 1 }

Note:

* In 'db.test1.createIndex( { "StudentId": 1 }, { unique: true } )', '"StudentId": 1' means the attribute is a part of the index. It does not mean the value of "StudentId" should be one. 1 stands for true here.
* In { unique: true }, the index is set to have the uniqueness property.

***Example:***

db.test1.insertMany([
   {   "StudentId" :2,
      "GPA": 3.72
   },
   {   "StudentId" :3,
      "GPA": 1.69
   },
   {
      "BCAssetId": "78c22fc6-5dec-11ec-bf63-0242ac130002",
      "BCAssetType": "BCAssetTypeMetadata",
      "BCAssetName": "BCAssetTypeMetadata: MBSEModel",
      "ForBCAssetType": "MBSEModel",
      "Version": {
         "Version": "1.0",
         "Subversion": null,
         "StartTime": "2019-01-13T07:23:13+06:00"
      }
   }
])
db.test1.find()

Note:

1. The method insertMany() inserts many documents.
2. Documents may have *no*schema.
3. Within a collection, there can be many kinds of documents.
4. StudentId is a unique index, but it may not exist.
5. Thus, a Mongo's unique index is not exactly the same as a candidate key (which cannot be null) of a table in the relational model.

**4. Querying**

* Basically use the find method.
* find as supported in Mongosh: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>.
* Format: db.collection.find(query, projection).

**4.1 Toyu**

Create the ‘toyu’ database in MongoDB.

1. Download the file: [toyu-db.gz](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/nosql/toyu-db.gz).
2. Ensure that you have download MongoDB tools: command line utilities including import and export, <https://www.mongodb.com/try/download/database-tools>.
3. Run the command:

mongorestore --archive="toyu-db.gz" --gzip --nsFrom='toyu.\*' --nsTo='toyu.\*'

Note that the design of toyu is not the typical way one would design a MongoDB. Instead, it is intended to look like the toyu MySQL database for ease of comparison.

***Example:***

[1] Show all students.

use toyu
db.student.find()

Getting rid of \_id:

db.student.find({},
   { "\_id": 0 }
)

[2] // Show all information of students majoring in 'CINF'.

db.student.find({"major": "CINF"},
    { "\_id": 0 }
)

[3] Show all student names. Return an array of student objects.

db.student.find({},
   { "fname": 1, "lname":1, "\_id": 0 }
)

[4] Show all student names in this format:

student #0: Tony Hawk
student #1: Mary Hawk
student #2: David Hawk
student #3: Catherine Lim
student #4: Larry Johnson
student #5: Linda Johnson
student #6: Lillian Johnson
student #7: Ben Zico
student #8: Bill Ching
student #9: Linda King

Solution:

result = db.student.find({},
   { "fname": 1, "lname":1, "\_id": 0 }
).toArray()

// May not always work as toArray() returns a promise,
// which may not be ready for use.
result.forEach((x,i) => console.log('student #' + String(i) + ': ' + x["fname"] + ' ' + x["lname"]))

[5] Show the names and credits (ach) of students majoring in 'CSCI' and having 40 or more credits.

db.student.find(
   { "major": "CSCI", "ach" : {$gte: 40} },
   { "fname": 1, "lname":1, "ach":1, "\_id": 0 }
)

Notes:

1. MongoDb's query and projection operators: <https://docs.mongodb.com/manual/reference/operator/query/>

[6] Show the first name and last name of students with a first name starting with a L or B, case insensitive.

db.student.find(
   { "fname": { $regex: /^[lb]/, $options: "i" } },
   { "fname": 1, "lname":1, "\_id": 0 }
)

Notes:

1. A regular expression is used: <https://docs.mongodb.com/manual/reference/operator/query/regex/#mongodb-query-op.-regex>.
2. For regular expressions in general, see: <https://en.wikipedia.org/wiki/Regular_expression>
3. Explanations:
	1. ^: match the beginning of a string.
	2. [lb]: a character class that matches 'l', 'b' (and also 'L' and 'B' since case insensitive matching is used.)
	3. option a: case insensitive matching.

[7] Show the names and credits (ach) of students majoring in 'CSCI' and having 40 or more credits.

db.student.find(
   { "$and": [ { "major": "CSCI"}, { "ach": {"$gte": 40}} ] },
   { "fname": 1, "lname":1, "ach":1, "\_id": 0 }
)

**4.2 Aggregation**

1. "Aggregation operations process multiple documents and return computed results."
2. See: <https://docs.mongodb.com/manual/aggregation/>.
3. It can be used to replace map-reduce functionality. See: <https://docs.mongodb.com/manual/reference/map-reduce-to-aggregation-pipeline/>.
4. There will not be programming questions on aggregation in the final examination.

[8] Show the number of faculty in each department.

In SQL:

SELECT DISTINCT deptCode, Count(facId)
FROM faculty
GROUP BY deptCode;

In MongoDB:

db.faculty.aggregate([
    {"$group" : {\_id:"$deptCode", "count":{$sum:1}}}
])

db.faculty.aggregate(
   [
      { $group: { "\_id": "$deptCode", "count": {$sum:1}} },
      { $project: { "deptCode": "$\_id" , "num\_faculty": "$count",  "\_id": 0}}
   ]
)

Notes:

1. $group: form group.
2. $sum: aggregate function.

[9] Show the names of students who have enrolled in 10000: joining two document.

This should have the similar effect of the SQL statement:

SELECT DISTINCT s.fname, s.lname
FROM student AS s, enroll AS e
WHERE s.stuId = e.stuId AND e.classId = 10000;

In MongoDB:

db.student.aggregate([
{$lookup:
    {
      from: "enroll",
      let: {joinValue: '$stuId'},
      pipeline: [
           { $match:
                 { $expr:
                    { $and:
                       [
                         { $eq: [ "$stuId",  "$$joinValue" ] },
                         { $eq: [ "$classId", 10000 ] }
                       ]
                    }
                 }
            }
        ],
        as: "enrollment"     }},
  { $match: {"enrollment":  { $ne: [] }}},
  { $project: { "fname": 1, "lname": 1, "\_id": 0}}
])

Notes:

1. An 'join' example.
2. Joining is difficult in MongoDB than SQL as document database should not be designed like a relational database.
3. In particular:
	1. The relational model uses a flat structure with no embedment.
	2. The document model uses a hierarchical structure encouraging embedment.

**4.3 Running Javascript program not using mongosh**

Try run [tinker.js.txt](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/nosql/tinker.js.txt) (remove .txt when saving)

// run "npm i mongodb" in the working directory.

// To run this program: node tinker1.js
const mongo = require('mongodb');

var MongoClient = mongo.MongoClient;
var url = 'mongodb://localhost:27017';

MongoClient.connect(url, function(err, client) {
   db = client.db("toyu");
   console.log("hello");
   var result = db.collection("faculty").find(
      { "rank": "Assistant Professor" },
      { "fname": 1, "lname": 1, "deptCode": 1, "\_id": 0,  }
   ).toArray()
   result.then((docs) => {
        console.log(docs);
    }).catch((err) => {
        console.log(err);
    }).finally(() => {
        client.close();
    });
});