Mongo
Node.js

More Mongo Queries

Querying Embedded Documents

_id		the mongo id	
actor	int	the id number given by sakila	
first_name	String	first name (all caps)	
last_name	String	last name (all caps)	
films	array	an array of embedded documents each of which has the following fields	
		categ	integer
		catname	string
		filmid	integer
		filmname	string

```
{
    first_name: 'ANNE',
    last_name: 'CRONYN',
    films: [
      {
        categ: 1,
        catname: 'Action',
        filmid: 927,
        filmname: 'UPRISING UPTOWN'
      },
      {
        categ: 1,
        catname: 'Action',
        filmid: 250,
        filmname: 'DRAGON SQUAD'
      }]
}
```

- Find info about actors in film with id 927
- db.xxx.find({"films.filmid": 927})

Quotes required

- find info about actors named DAVE
- db.xxx.find({first_name: "DAVE"})

Quotes optional

Projections

Selecting fields to return

- db.collection.find(selection, projection)
- defaults is to return everything
- So projection really says what NOT to return
  - {fieldname:0, fieldname:0, ...}

- db.XXX.find({"first_name": "BOB"}, {"films.categ":0, "films.catname":0, _id:0, actor:0})

Queries

command	explanation
$lt	<
$lte	<=
$gt	>
$gte	>=
$ne	≠
$exists	a field exists
$in	{ field: { $in: [value1, value2, ...] } }
$nin	
$and $or	
$regex	match using a regular expression

find one actor whose actorid is less than 5

find all actors whose actorid is less than or equal to 5

find all actors whose actor id is greater than 198

find all actors in films with an id greater than or equal to 990

find all actors whose name is not BOB

find all actors whose name is BOB or LUCILLE (use in)

find all actors whose name is not BOB or LUCILLE

find all actors whose name is BOB or LUCILLE (use or)

find all actors whose name begins with D

See today's lab (9) for answers

Node.js

Install the Mongo driver

- (as with everything else) Do everything on 165.106.10.133
  - the mongo server will not talk to processes running anywhere else

- UNIX> npm install mongodb@3.7
  - using an **old** version of node so need an **old** version of the mongo driver

  - There is an alternative npm package "mongoose"
    - Does schema validation

Connect to Mongo

```
const { MongoClient } = require('mongodb');

const uri = "mongodb://127.0.0.1/sakila";
const client = new MongoClient(uri, { useUnifiedTopology: true });
```

After doing the npm install

No login or authentication.
Connect to sakila database

Get the connection set up

Just avoid some annoying messages

async and await

- "await" can only be used inside functions declared to be "async"
- "await" waits for the called function to finish
  - the called function must itself be declared to be "async".

```
async function dooer() {
    console.log(`AAA ${new Date()}`);
    let val = await somethingSlow();
    console.log(`BBB ${new Date()} ${aaa}`);
}

console.log(`YYY ${new Date()}`);
dooer();
console.log(`ZZZ ${new Date()}`);
```

Connect to Mongo

Using node without the bother of making a website

```
const { MongoClient } = require('mongodb');

const uri = "mongodb://127.0.0.1/sakila";
const client = new MongoClient(uri, { useUnifiedTopology: true });


async function main(){

    try {
        await client.connect();
    } catch (e) {
        console.error(e);
    } finally {
        await client.close();
    }
}

console.log(`YYY ${new Date()}`);
await main();
console.log(`ZZZ ${new Date()}`);
```

to run:
node xxx.js

Actually connect

Destroy connection
Do NOT do this if using express

Querying Mongo

- Queries look a lot like those from mongosh
- toArray() gets all the query results and puts them into array rather than dealing with cursors
- Results are in array of Objects
  - each Object has the structure of the retrieved document
  - Objects in javascript can be indexed much like arrays

```
const { MongoClient } = require('mongodb');
const uri = "mongodb://127.0.0.1/sakila";
const client = new MongoClient(uri, { useUnifiedTopology: true });

async function main(){
   try {
        await client.connect();
        await doQueryA(client);
    } catch (e) {
        console.error(e);
    } finally {
        await client.close();
        console.log("Really done main");
    }
}

async function doQueryA(client) {
    let spec = { first_name: { $regex: "^N" } }
    let results = await client.db().collection('mgoactor').find(spec).toArray();
    for (let d = 0; d < results.length; d++) {
        console.log(`QA ${results[d]['first_name']} ${results[d]['last_name']}` );
    }
    //console.log(results);
}

main();
```

Regular expressions
"starts with N"

more on find

db.collection.find(<query>, <projection>, <options>)

- Each arg is a "document"
  - a JSON object
- Query is exactly what you would put into "findOne" or "findMany" in mongosh
  - insertion attacks?
- Projection
  - {projection: {field1: 1, field2: 0, ...} }
    - 1 == get field
    - 0 == do not get field
    - handling embedded using dot notation
- Options
  - There are many https://mongodb.github.io/node-mongodb-native/4.0//interfaces/findoptions.html

Without "projection" this all gets ignored

In the version of node-mongo connector that we are using cannot specify "inclusion"

Better Query Function

- Projections require some annoying extra syntax
- Then, for grins, pass in a function to format up the results.
  - As opposed to returning the results and passing return to a renderer

```
    // in some async function
    await doQuery(client, 'mgoactor',
                          { first_name: { $regex: "^N" } },
                          { projection: { _id: 0 } },
                          simpleRenderer);

function simpleRenderer(results) {
    for (let d = 0; d < results.length; d++) {
        console.log(`SimpleRenderer ${results[d]['first_name']} ${results[d]['last_name']}` );
    }
}

// a more general querying system.
async function doQuery(client, collection, spec, restrict, renderer) {
    let results = await client.db().collection(collection).find(spec, restrict).toArray();
    if (renderer != null) {
        renderer(results)
    }
}
```

Node middleware for Mongo

- Repeatedly asking to connect to mongo is OK
- By default mongo uses a connection pool with 100 connections max
- Returns a "table" in the stupidest possible way

```
const { MongoClient } = require('mongodb');
const express = require('express')

const uri = "mongodb://127.0.0.1/sakila";
const client = new MongoClient(uri, { maxPoolSize: 5, useUnifiedTopology: true } );
const port = 30001
const app = express()

let cc = 0

async function main(req, resp, letter){

    try {
        await client.connect();
        await doQueryA(client, req, resp, letter);
   } catch (e) {
        console.error(e);
   }}

async function doQueryA(client, req, resp, letter) {
    resp.write("<html><body><pre>")
    let spec = { first_name: { $regex: letter } }
    let results = await client.db().collection('mgoactor').find(spec).toArray();
    for (let d = 0; d < results.length; d++) {
        console.log(`QA ${results[d]['first_name']} ${results[d]['last_name']}`)
        resp.write(`QA ${results[d]['first_name']} ${results[d]['last_name']}\n`)
    }
    resp.end("</pre></body></html>")
}

app.use('/mongo', function (req, res) {
    main(req, res, "^N");
});

app.listen(port, function(error){
    if(error) throw error
})
```

Better node middleware

- Connect once on startup
- Return JSON
- take the first letter to be searched for as parameter in JSON submission
- Use projection to limit returned data

```
const { MongoClient } = require('mongodb');
const express = require('express')
const bodyParser = require('body-parser')

const uri = "mongodb://127.0.0.1/sakila";
const client = new MongoClient(uri, { maxPoolSize: 5,
useUnifiedTopology: true } );
const port = 30001
const app = express()
app.use(bodyParser.json());

let cc = 0

async function connectMongo() {
    await client.connect();
    console.log(`Connected to Mongo!! ${cc++}`)
}

async function doQueryA(req, resp, letter) {
    let spec = { first_name: { $regex: letter } }
    let results = await
client.db().collection('mgoactor').find(spec, { projection:
{ "films": 0, "_id": 0 } }).toArray();
    resp.end(JSON.stringify(results))
}


app.post('/actors', function (req, res) {
    doQueryA(req, res, req.body['letter']);
});


app.listen(port, function(error){
    if (error) throw error
    connectMongo()
    console.log("Server created Successfully")
})
```

References

- Connecting Mongo to Node.js
  - https://www.mongodb.com/blog/post/quick-start-nodejs-mongodb-how-to-get-connected-to-your-database
- Mongo CRUD operations in Node.js
  - https://www.mongodb.com/developer/quickstart/node-crud-tutorial/?_ga=2.248826372.220238191.1650160314-1645336960.1646679840
- Projections syntax in Node.js
  - https://stackoverflow.com/questions/47732061/node-js-mongodb-find-with-projection-to-exclude-id-still-returns-it

Mongo and Python

PyMongo

- PyMongo is the python package for writing to mongo dbs
- Generally the syntax closely mirrors that of mongosh

```
>>> import pymongo
>>> from pymongo import MongoClient
>>> client = MongoClient()
>>> db = client.sakila
>>> db.list_collection_names()
['city', 'inventory', 'mgofilm', 'staff', 'mgoactor', 'language', 'film_actor',
'rental', 'film_category', 'film', 'country', 'actor', 'store', 'address',
'payment', 'customer', 'category']
>>> db.mgoactor.find_one()
{'_id': ObjectId('6240ec924ff1d4bc11fa3603'), 'actor': 1, 'first_name':
'PENELOPE', 'last_name': 'GUINESS', 'films': [{'categ': 2, 'catname':
'Animation', 'filmid': 23, 'filmname': 'ANACONDA CONFESSIONS'}, {'categ': 3,
'catname': 'Children', 'filmid': 509, 'filmname': 'LANGUAGE COWBOY'},... ]}
```

pymongo basics

- • Python Dictionaries are used to represent documents
- • Once you instantiate a collection in python, you can do lots of simple things
  - ○ my_collection.find_one()
  - ○ my_collection.insert_one()
  - ○ my_collection.insert_many()
  - ○ my_collection.count_documents({})
  - • my_collection.find() returns a Cursor object, which is iterable, and contains documents

```
>>> db = client.sakila
>>> collection = db.mgoactor
```

PostgreSQL to Mongo

Table == collection
(which is generally wrong)

```
import pymongo
from pymongo import MongoClient

import psycopg2
#from psycopg2.extras import RealDictCursor

import json
from datetime import date, datetime


CONNECTION_STRING = "mongodb://127.0.0.1:27017/sakila"
client = MongoClient(CONNECTION_STRING)
rdb = client['sakila']

cnx = psycopg2.connect(user="dbuser", database="sakila", host="localhost", port="5432", password="12345678")
cursor = cnx.cursor()

tables = ['actor', 'address', 'category', 'city', 'country', 'customer', 'film', 'film_actor', 'film_category', 'inventory',
'language', 'payment', 'rental', 'staff', 'store']

for table in tables:
    scoll = rdb[table]
    print(table)
    cursor.execute("with aaa as (select * from {}) select json_agg(t) from aaa as t;".format(table))
    results = cursor.fetchall()
    if len(results) > 0:
        #print(results[0][0])
        scoll.insert_many(results[0][0])
```

PostgreSQL to Mongo

more thoughtfully

- Rather than just copying table to collection
  - make major merges to avoid joins
  - accept data duplication

```
from pymongo import MongoClient
import psycopg2
import json
from datetime import date, datetime
CONNECTION_STRING = "mongodb://127.0.0.1:27017/sakila"
client = MongoClient(CONNECTION_STRING)
rdb = client['sakila']
cnx = psycopg2.connect(user="dbuser", database="sakila",
host="localhost", port="5432", password="12345678")
cursor = cnx.cursor()
def addNames(aid, actors, mapp):
    for aa in actors:
        #print(aa)
        if aa[0] == aid:
            mapp['first_name'] = aa[1]
            mapp['last_name'] = aa[2]

def getCatName(cid, categories):

def getFilmName(fid, films):

def getFilm(fid, films):

def getLanguageName(lid, langs):
```

```
rrr = []
th = -1
cur = []
for r in results:
    if th!=r[0]:
        #print(cur)
        if th>-1:
            ss={}
            ss['actor'] = th
            addNames(th, resultsA, ss);
            ss['films'] = cur
            rrr.append(ss)
        th=r[0]
        cur=[]
    for f in r[1]:
        r4 = {}
        r4['categ'] = r[2]
        r4['catname'] = getCatName(r[2], resultsC)
        film=getFilm(f, resultsF)
        r4['filmid'] = f
        if film!=None:
            r4['filmname'] = film[1]
        cur.append(r4)
    #rr['categ']=r[2]
    #rr['films']=r[1]
    #cur.append(rr)

if len(cur)>0:
    ss['actor'] = th
    ss['films'] = cur

#print(json.dumps(rrr[0:5]))
scoll = rdb['mgoactor']
scoll.insert_many(rrr)
```

Learn more about PyMongo:
https://pymongo.readthedocs.io/en/stable/tutorial.html