Mongo & Node.js

Install the Mongo driver

- Do everything on 165.106.10.170
  - You can set up tunnels and do things from your local machine but setting up the tunnel is probably not worth the savings

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

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

Just avoid some annoying messages

async and await

- unlike postgres in which you pass a function that asynchronously gets the result of a query, the norm with Mongo is to use "async" functions 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

With await and async

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

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


async function main(){

    try {
        console.log(`AAA ${new Date()}`);
        await client.connect();
        console.log(`BBB ${new Date()}`);
    } catch (e) {
        console.error(e);
    } finally {
        console.log(`CCC ${new Date()}`);
        await client.close();
        console.log(`DDD ${new Date()}`);
    }
}

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

Actually connect

Destroy connection

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 looks exactly like the given collections structure
  - 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);
}
```

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: { first_name: 1, last_name:1, _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)
    }
}
```

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

Problem: Phrases
Solution: Positional indexes

- In the inverted inded, store, for each *term* the position(s) in which it appears:

  - <***term***, number of docs containing ***term***;
  - *doc1*: position1, position2 … ;
  - *doc2*: position1, position2 … ;
  - etc.>

For Computing IDF

Positional Index Example

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

Which of docs 1,2,4,5 could contain "***to be or not to be***"?

Processing a Phrase Query

- Extract inverted index entries for each distinct term: *to, be, or, not.*
- Merge their *doc:position* lists to enumerate all positions with "*to be or not to be*".
  - *to:*
    - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - *be:*
    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...

- Same general method for proximity searches
  - "LIMIT! /3 STATUTE /3 FEDERAL /2 TORT"
    - /*k* means "within *k* words of"
  - Positional indexes can be used for such queries; phrase indexes cannot.

Positional Index Size

TINFL

1. A positional index expands postings storage *substantially*
    1. Even though indices can be compressed
    2. Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries
2. Need an entry for each occurrence, not just once per document
    1. Index size depends on average document size and average frequency of each term
        1. Average web page has <1000 terms
        2. SEC filings, books, even some epic poems … easily 100,000 terms
3. Rule of Thumb
    1. A positional index is 2–4 as large as a non-positional index
    2. Positional index size 35–50% of volume of original text