Databases and MongoDB

Michael Chang

Spring 2023

Plan for today

Intro to databases

- Types of databases, schemas

Intro to MongoDB

- The MongoDB shell

MongoDB with Node

- Integrating with our REST API

BTW, install MongoDB

If you want to follow along today

- You'll need to install MongoDB
- (Will have to do this for assign3.2 anyway)
- https://cs193x.stanford.edu/mongodb.html

Types of databases

SQL databases

- Traditional database; stores data in tables
- Each table has fields (columns) and records (rows)
- Fields can relate (refer) to each other
  - E.g. students have advisors, advisors belong to departments

students

id	name	advisorId
1	Kashif	3
2	Michael	4

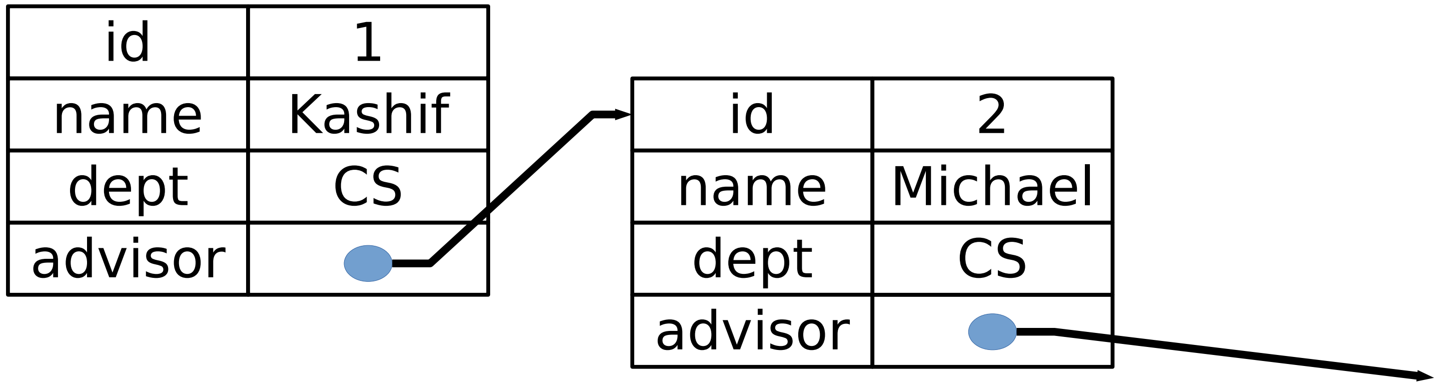
advisors

id	name	deptId
3	Michael	CS
4	Julie	CS

Types of databases

NoSQL

- Reaction to rigid structure of SQL databases
- Stores data as documents
- Documents can refer to each other
  - (But we won't go into this part)



MongoDB

- **A NoSQL database server**
  - Listens for connections on a port
  - Clients connect to store and retrieve data
  - Heavy integration with JavaScript
    - This makes it easy to integrate into our backend
- **Structure**
  - Server has multiple databases
  - Each database has multiple collections
  - Each collection stores documents
  - Documents are just JavaScript objects

mongosh shell

Run `mongosh` for interactive access

```
$ mongosh
> use example
> db.myCollection.insertOne({
... id: 1,
... name: "Michael"
})
```

(There used to be another command called `mongo` but they removed that)

mongosh shell

`show dbs`: list databases

`use <db>`: switch databases

`show collections`: list collections

`db.<collection>.<operation>(<args>)`

- JavaScript method calls

- Databases and collections created automatically when first document inserted

collection operations

`.insertOne(doc)`: insert document

- doc is a document (JavaScript object)

`.insertMany(docs)`: insert multiple

- <docs> is an array of documents

`.find([query])`: retrieve documents

- With no query, get all documents
- query is a document; finds documents with matching key/values
  - E.g. `db.students.find({ id: "michael" })`

`.findOne([query])`: retrieve first document

- (Mostly) no guaranteed order

Aside: _id field

All documents automatically get an _id

- Unique across all documents in a collection
- You can use it to look up documents if you want
- But for simplicity we will use our own unique identifiers
- Don't confuse this with "id", which is not special

Query operators

Use query operators for complex searches

- Operators start with $

`$gt, $lt, $gte, $lte`: comparison

- E.g. `db.courses.find({ units: {$gte: 3} })`

`$in`: matches element in array

- E.g. `db.students.find({ id: {$in: ["kashif", "michael"]} })`

`$regex`: match text (regular expression)

- E.g. `db.courses.find({ code: {$regex: "^CS106"} })`

Update and delete

`.replaceOne(query, doc)`

Replace first document matching query with doc

`.deleteOne(query)`

`.deleteMany(query)`

Delete document(s) matching query

`.updateOne(query, update)`

`.updateMany(query, update)`

Apply update operations to document(s) matching query

Update operators

- **Update operators let you change documents**
- **$set: set key/values**
  - E.g.
    ```
    db.students.updateOne(
      { id: "kashif" },
      { $set: { advisor: "mchang" } }
    )
    ```
- **There's a bunch of others**
  - You can use them if you want, but for our needs, fine to just replace
- **Don't use update when you want replace**
  - You'll get confusing errors about not using update operators

Misc

`db.<collection>.drop()`

Delete collection

`db.dropDatabase()`

Delete the database

mongodb in Node

`npm install mongodb`

- Library for connecting to MongoDB server
- Most method names same as shell
- A few differences for setup/connection

Callback and Promise interface

- Most functions take callbacks
- If no callback, returns a Promise
- We'll exclusively use Promise interface (with await)

Connecting to MongoDB

```
import { MongoClient } from "mongodb";
const main = async () => {
  let conn = await
    MongoClient.connect("mongodb://127.0.0.1");
};
```

`MongoClient`

Class for connecting to the Mongo server

`MongoClient.connect(url, options)`

url: host and port of server, starts with mongodb://

`127.0.0.1` is the same as `localhost`, but MongoDB doesn't always handle localhost correctly

Accessing a collection

```
let db = conn.db("myDatabase");
let Students = db.collection("students");
await Students.insertOne({ ... });
```

`conn.db(name)`

Get a database object (not async)

`db.collection(name)`

Get collection object (not async)

Collection

This documentation is a bit rough

Can use it to see list of methods

Parameters are mostly the same as the mongosh commands

MongoDB in Node

Most methods are the same as in mongosh

- `.insertOne`, `.insertMany`, `.replaceOne`, `.deleteMany`, ...
- But they are all async

`findOne` largely the same

- Returns matching document, or null

`find` returns a "Cursor"

- Can use `.hasNext()` and `.next()` to loop through
- Or use `.toArray()` to convert to array (easier)
- `find` not async, `hasNext`/`next`/`toArray` are

```
let docs = await Students.find().toArray();
```

Aside: CORS

Normally can't fetch() from different "origin"

- Origin = host and port
- E.g. if server running on another machine, or another port on same machine

  ```
  fetch("http://localhost:1931/api");
  ```

Cause: CORS

- Prevents malicious web sites from reading content from your pages/APIs

Solution

```
import cors from "cors";
app.use(cors());
```

Summary

This week

- Backends and databases
- Can now build simple but powerful REST APIs

Before next time

- assign3.1
- Will post assign3.2 tomorrow (due next Thu)
- Will post more project info over weekend

Next week

- Full stack topics
- E.g. authentication, mobile, accessibility, CSS animations