Databases and MongoDB

Michael Chang

Spring 2020

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 assign4 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

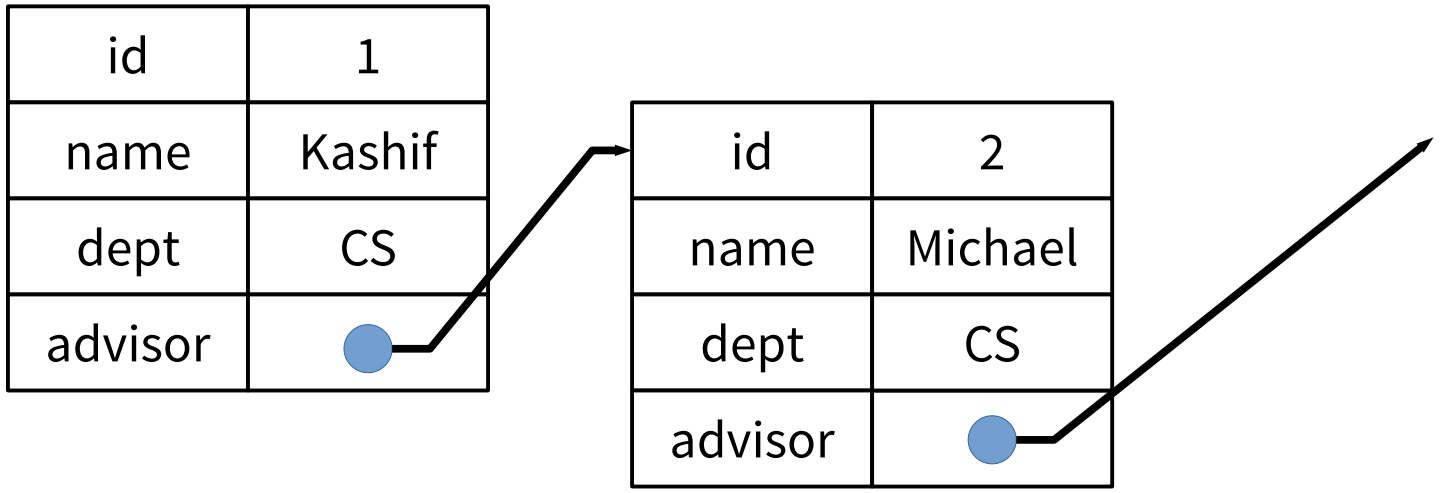
NoSQL

React 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 and 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

mongo shell

Run `mongo` for interactive access

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

mongo 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: "michael "} }
)
```

There's a bunch of others

You can use them if you want, but for our needs, fine to just replace

Don't mix update and replace

You'll get confusing errors

Misc

`db.<collection>.drop()`

Delete collection

`db.dropDatabase()`

Delete the database

So far

Intro to databases

Types of databases, schemas

Intro to MongoDB

The MongoDB shell

MongoDB with Node

Integrating with our REST API

mongodb in Node

`npm install mongodb`

- Library for connecting to MongoDB server
- Most methods and documents same as shell
- A few differences for setup

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

```
const { MongoClient } = require("mongodb");
const main = async () => {
  let conn = await MongoClient.connect(
    "mongodb://localhost",
    { useUnifiedTopology: true }
  );
};
```

`MongoClient`

Class for connecting to the Mongo server

Connecting to MongoDB

```
const { MongoClient } = require("mongodb");
const main = async () => {
  let conn = await MongoClient.connect(
    "mongodb://localhost",
    { useUnifiedTopology: true }
  );
};
```

MongoClient.connect(url, options)

- url: host and port of server, starts with mongodb://
- useUnifiedTopology: this enables some new connection behavior
  - Without it, you get a deprecation warning
  - ...you can just add it and move on

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 methods

`insertOne`, `insertMany`, `replaceOne`, `deleteOne`, `deleteMany` all the same

find and findOne

`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 cursor = Students.find();
while (await cursor.hasNext())
  console.log(await cursor.next());
// Or...
let docs = await Students.find().toArray();
```

Summary

This week

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

assign4 out tonight

- Backend for assign3

Next week

- Full stack topics (haven't worked out order yet)
- E.g. authentication, mobile, accessibility, CSS animations