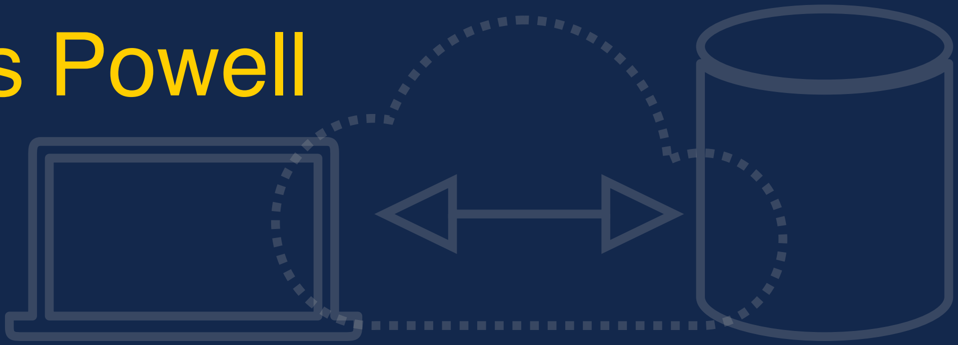
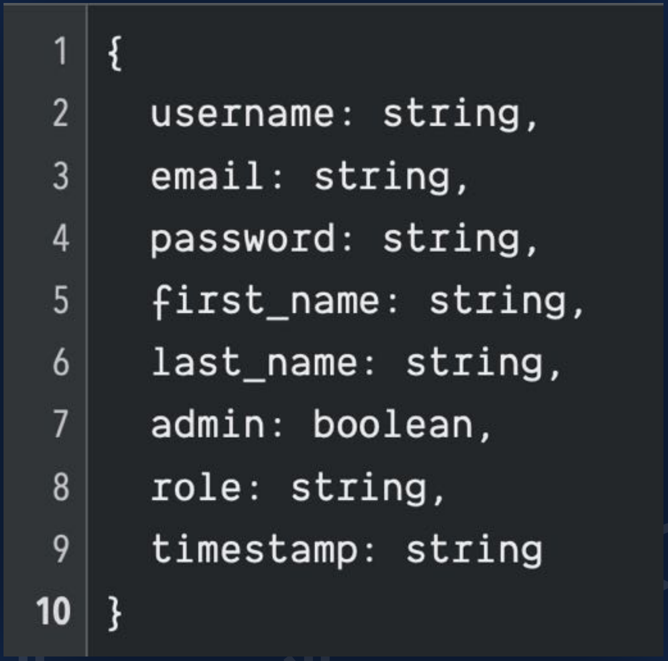
Database Driven Sites
Part II: MongoDB

with Thomas Powell



Step 1: Model Your Data

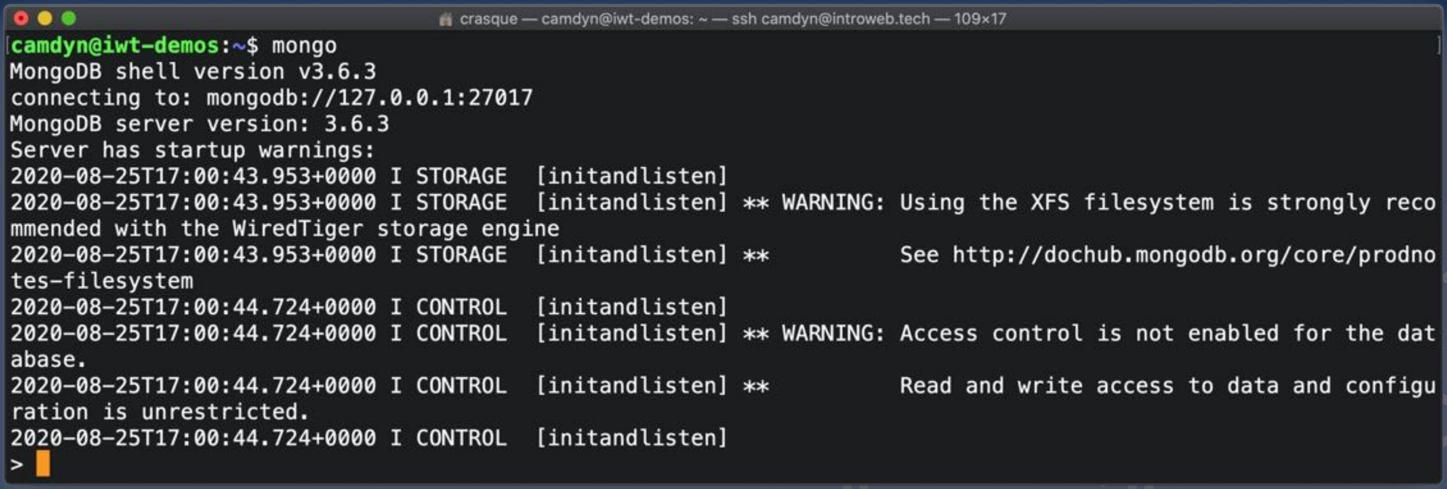
- Unlike a SQL style database, MongoBD is a non-structured database meaning you don’t explicitly create a schema for your collections.
- It may still prove useful though to model your data beforehand for your own organizational purposes.
- MongoDB uses JSON to structure it’s data



```
{
  username: string,
  email: string,
  password: string,
  first_name: string,
  last_name: string,
  admin: boolean,
  role: string,
  timestamp: string
}
```

Step 2 – Create the Database

- If you have shell access to a server you might use the mongo shell to admin the database
- mongo —host *hostname* -u *username* -p *password*
  - If you're just trying to access your local mongo, you can simply use the command **`mongo`**

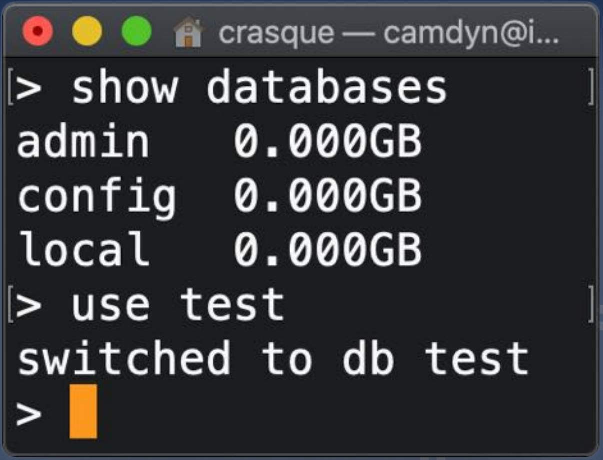

crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 109×17

```
camdyn@iwt-demos:~$ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
Server has startup warnings:
2020-08-25T17:00:43.953+0000 I STORAGE  [initandlisten]
2020-08-25T17:00:43.953+0000 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly reco
mmended with the WiredTiger storage engine
2020-08-25T17:00:43.953+0000 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodno
tes-filesystem
2020-08-25T17:00:44.724+0000 I CONTROL  [initandlisten]
2020-08-25T17:00:44.724+0000 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the dat
abase.
2020-08-25T17:00:44.724+0000 I CONTROL  [initandlisten] **          Read and write access to data and configu
ration is unrestricted.
2020-08-25T17:00:44.724+0000 I CONTROL  [initandlisten]
> 
```

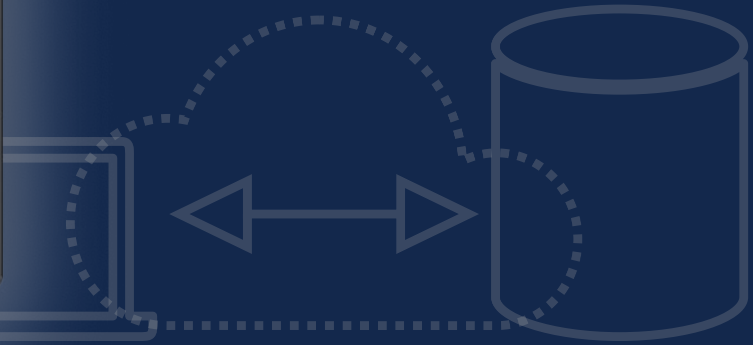
Step 2 – Create the Database

- In MongoDB, to view all of the databases you currently have, you simply use the command **`show databases`**

- Then, to create a new database in MongoDB it's the exact same as selecting any other database. If the database doesn't exist, it is simply created.

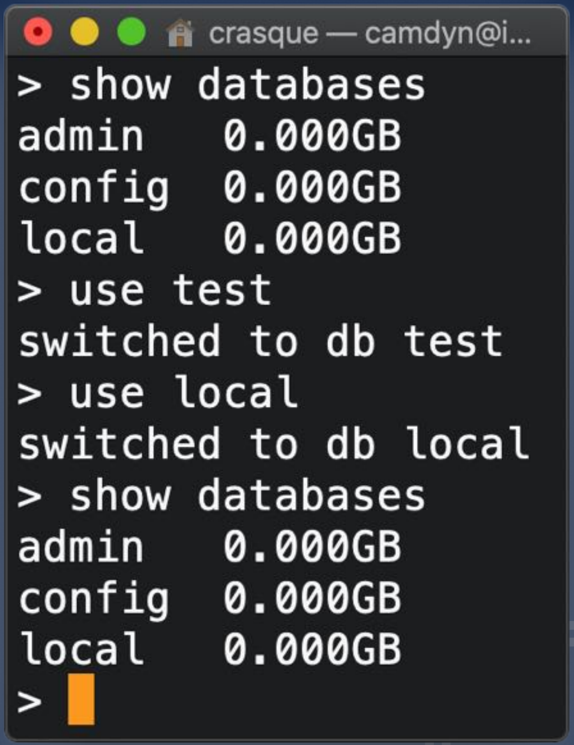


```
> show databases
admin    0.000GB
config   0.000GB
local    0.000GB
> use test
switched to db test
> 
```



Step 2 – Create the Database

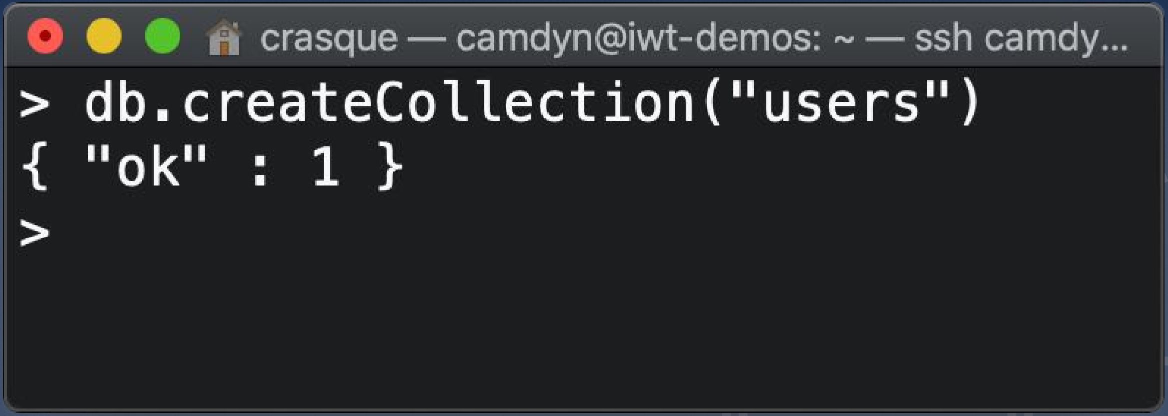
- It is important to note that in Mongo unless you save a document inside that database, it won’t be saved.



```
> show databases
admin   0.000GB
config  0.000GB
local   0.000GB
> use test
switched to db test
> use local
switched to db local
> show databases
admin   0.000GB
config  0.000GB
local   0.000GB
> 
```

Step 2 – Creating a Collection

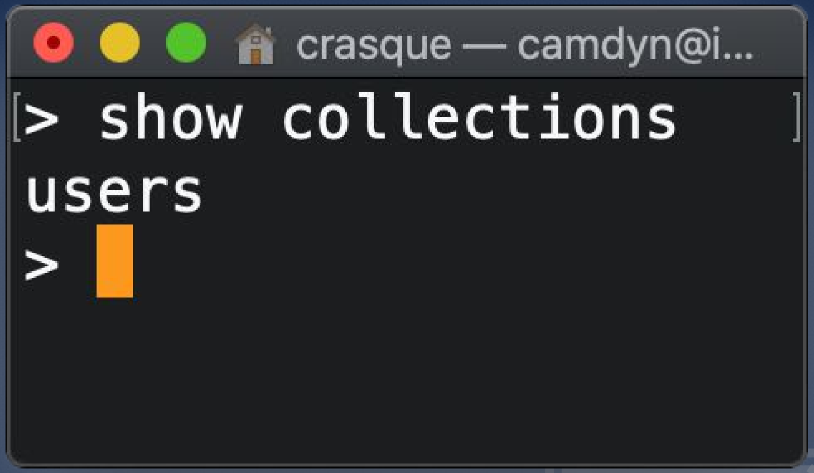
- The Mongo equivalent of a table is a collection. Similar to how the database was created, you can simply just start using a collection whether it exists or not and one will be created.
- If you would like to explicitly create a collection, you can do such like this:



```
> db.createCollection("users")
{ "ok" : 1 }
>
```

Viewing your Collections

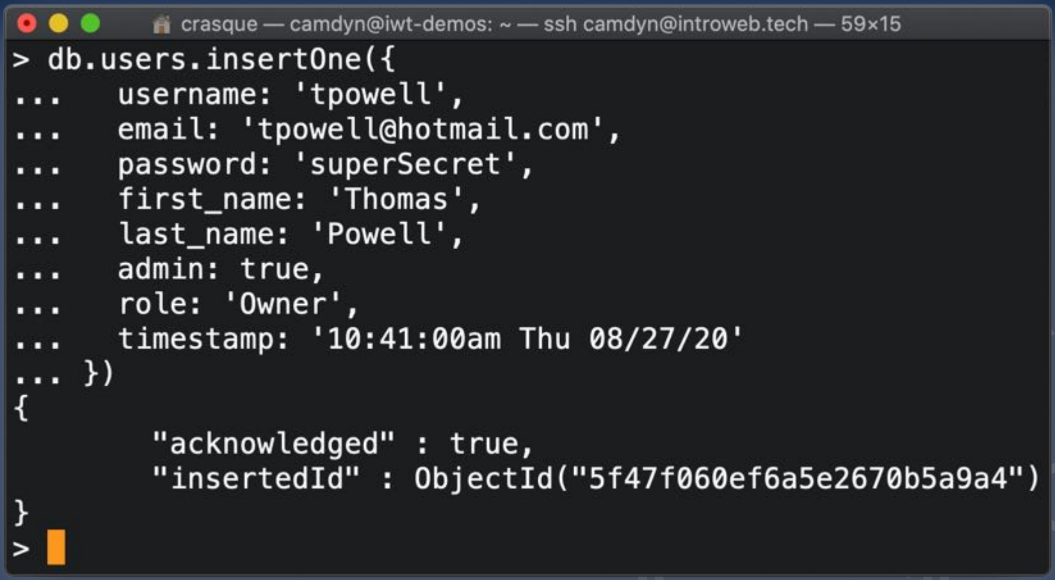
- To quickly view which collections you currently have, just like with databases, you use the command **`show collections`**



```
> show collections
users
> 
```

Inserting Data

- Where as you'd insert a row in SQL, in Mongo you'd insert a document. To insert a single document into a collection, you simply use the format `db.collection.insertOne()`.
- Again, whether the collection exists or not does not matter, as if it doesn't exist it will be created upon this insertion.

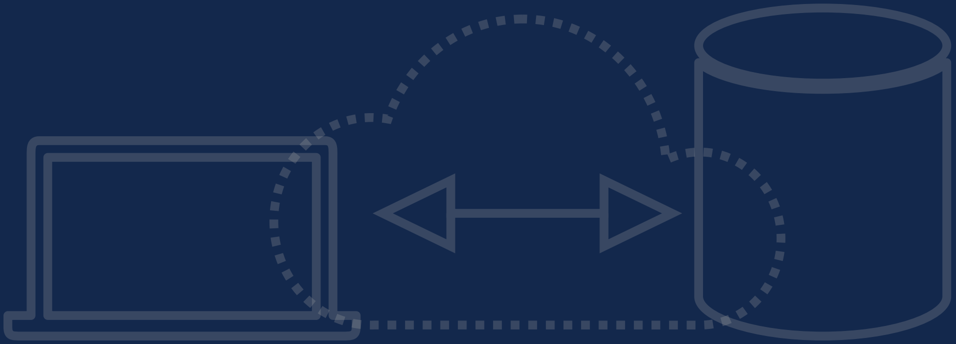

crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 59×15

```
> db.users.insertOne({
...   username: 'tpowell',
...   email: 'tpowell@hotmail.com',
...   password: 'superSecret',
...   first_name: 'Thomas',
...   last_name: 'Powell',
...   admin: true,
...   role: 'Owner',
...   timestamp: '10:41:00am Thu 08/27/20'
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5f47f060ef6a5e2670b5a9a4")
}
> 
```

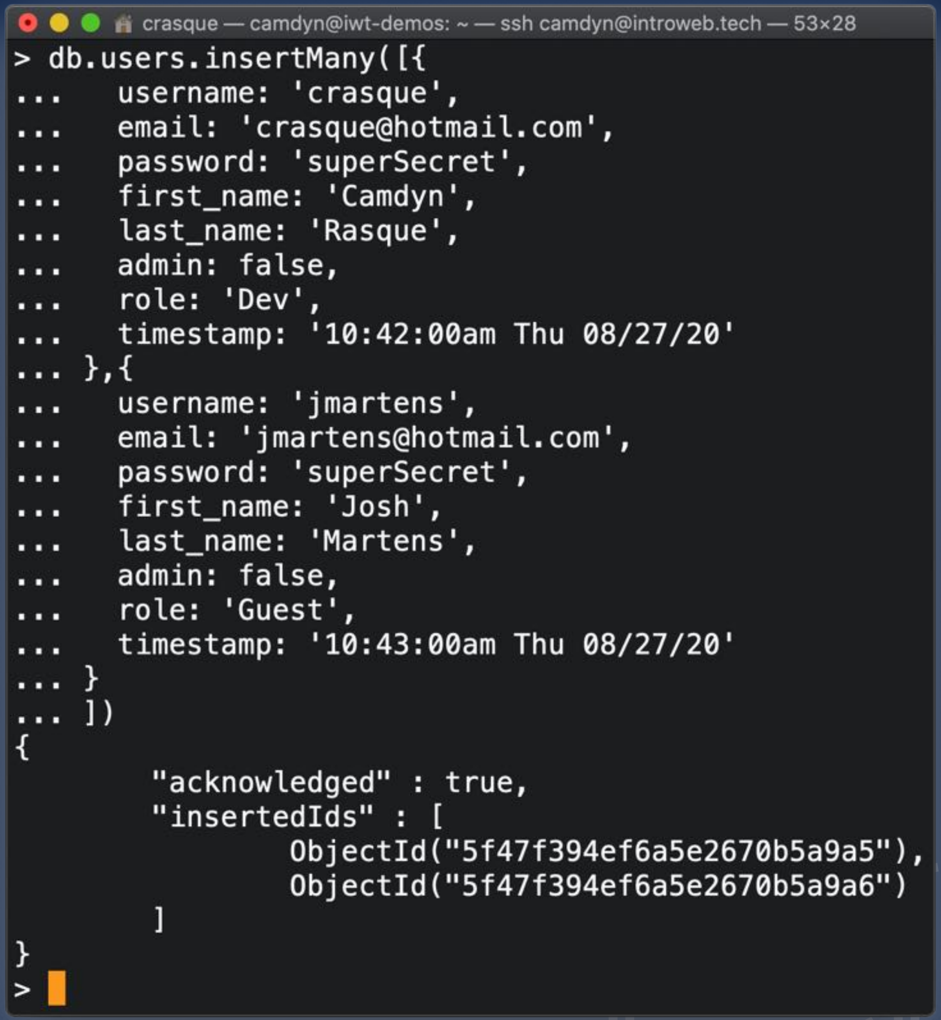
Inserting Data

- As you might have guessed, to insert more than one document at a time you simply use the command `db.collection.insertMany()`.

- The only difference in usage is that you just need to wrap your individual JSON objects in an array when you pass them in



Inserting Data


crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 53×28

```
> db.users.insertMany([{
...    username: 'crasque',
...    email: 'crasque@hotmail.com',
...    password: 'superSecret',
...    first_name: 'Camdyn',
...    last_name: 'Rasque',
...    admin: false,
...    role: 'Dev',
...    timestamp: '10:42:00am Thu 08/27/20'
... },{
...    username: 'jmartens',
...    email: 'jmartens@hotmail.com',
...    password: 'superSecret',
...    first_name: 'Josh',
...    last_name: 'Martens',
...    admin: false,
...    role: 'Guest',
...    timestamp: '10:43:00am Thu 08/27/20'
... }
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5f47f394ef6a5e2670b5a9a5"),
                ObjectId("5f47f394ef6a5e2670b5a9a6")
        ]
}
> 
```

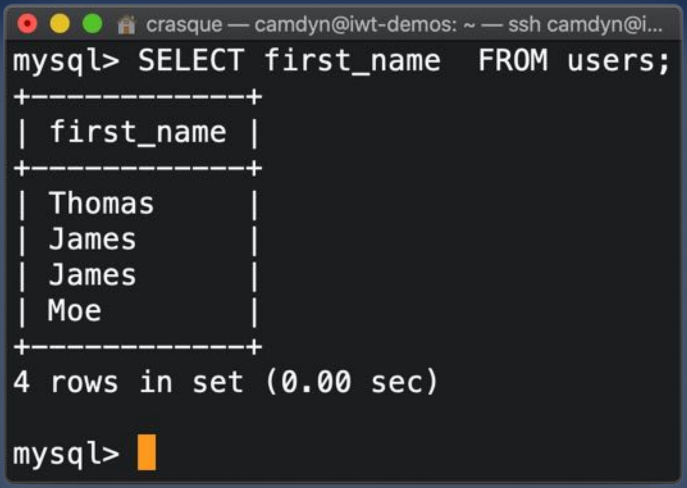
Finding Data

- Equivalent to SQL’s SELECT function, the mongo equivalent is `db.collection.find()`.

- It has two optional parameters: `Query` and `Projection` (both JSON objects).
  - `Query`: This is the field values/patterns you want to match to make your selection
  - `Projection`: This specifies which fields of the matched documents to return

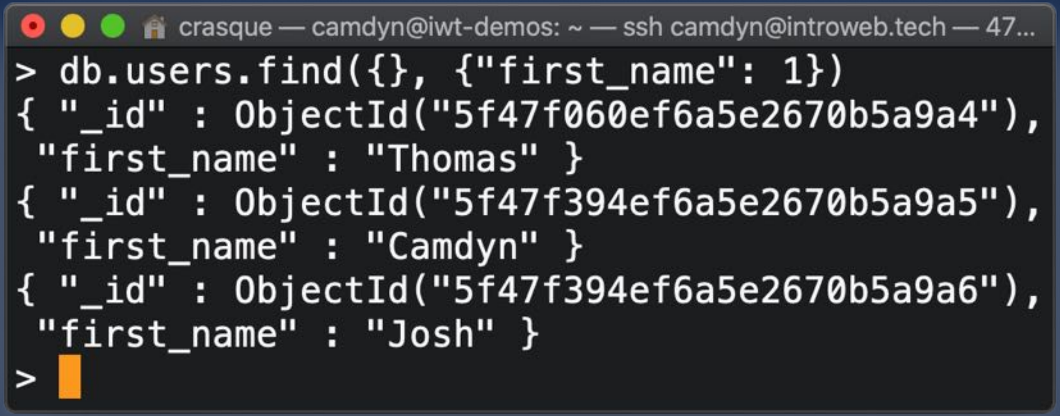
Finding Data

MySQL



```
mysql> SELECT first_name  FROM users;
+------------+
| first_name |
+------------+
| Thomas     |
| James      |
| James      |
| Moe        |
+------------+
4 rows in set (0.00 sec)

mysql> 
```

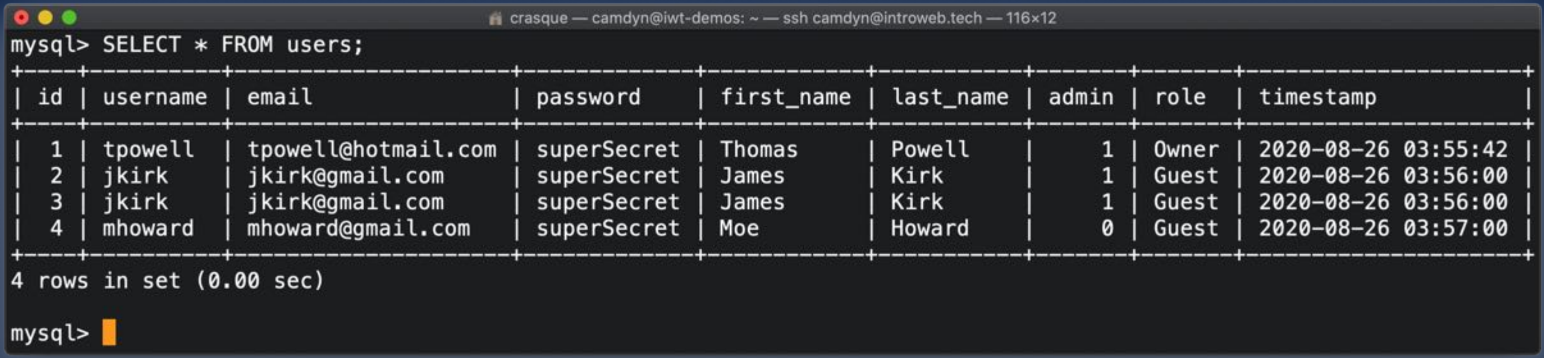
MongoDB


crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 47...

```
> db.users.find({}, {"first_name": 1})
{ "_id" : ObjectId("5f47f060ef6a5e2670b5a9a4"),
 "first_name" : "Thomas" }
{ "_id" : ObjectId("5f47f394ef6a5e2670b5a9a5"),
 "first_name" : "Camdyn" }
{ "_id" : ObjectId("5f47f394ef6a5e2670b5a9a6"),
 "first_name" : "Josh" }
> 
```

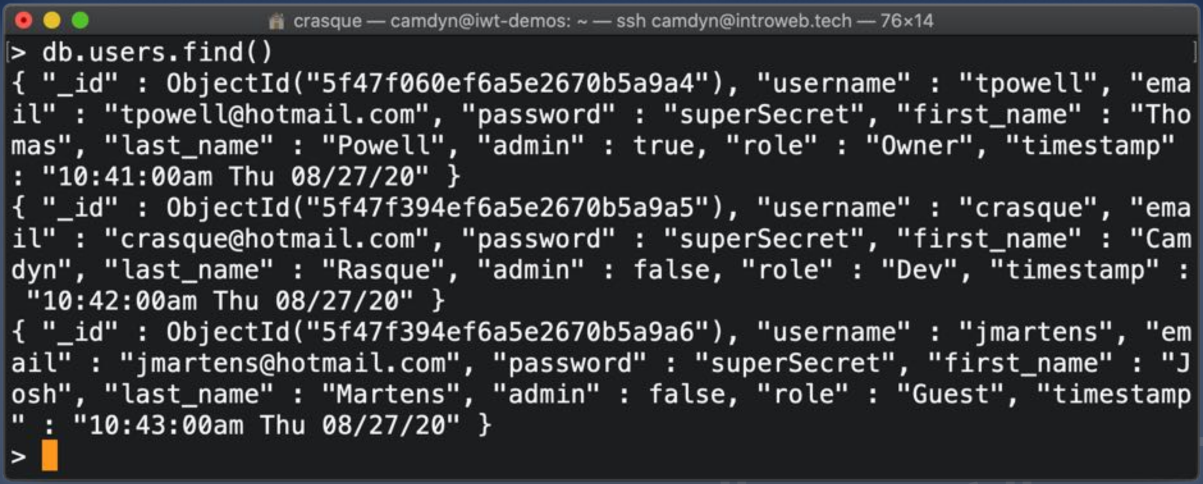
Selecting Data

MySQL


crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 116×12

```
mysql> SELECT * FROM users;
+----+----------+---------------------+-------------+------------+-----------+-------+-------+---------------------+
| id | username | email               | password    | first_name | last_name | admin | role  | timestamp           |
+----+----------+---------------------+-------------+------------+-----------+-------+-------+---------------------+
|  1 | tpowell  | tpowell@hotmail.com | superSecret | Thomas     | Powell    |     1 | Owner | 2020-08-26 03:55:42 |
|  2 | jkirk    | jkirk@gmail.com     | superSecret | James      | Kirk      |     1 | Guest | 2020-08-26 03:56:00 |
|  3 | jkirk    | jkirk@gmail.com     | superSecret | James      | Kirk      |     1 | Guest | 2020-08-26 03:56:00 |
|  4 | mhoward  | mhoward@gmail.com   | superSecret | Moe        | Howard    |     0 | Guest | 2020-08-26 03:57:00 |
+----+----------+---------------------+-------------+------------+-----------+-------+-------+---------------------+
4 rows in set (0.00 sec)

mysql> 
```

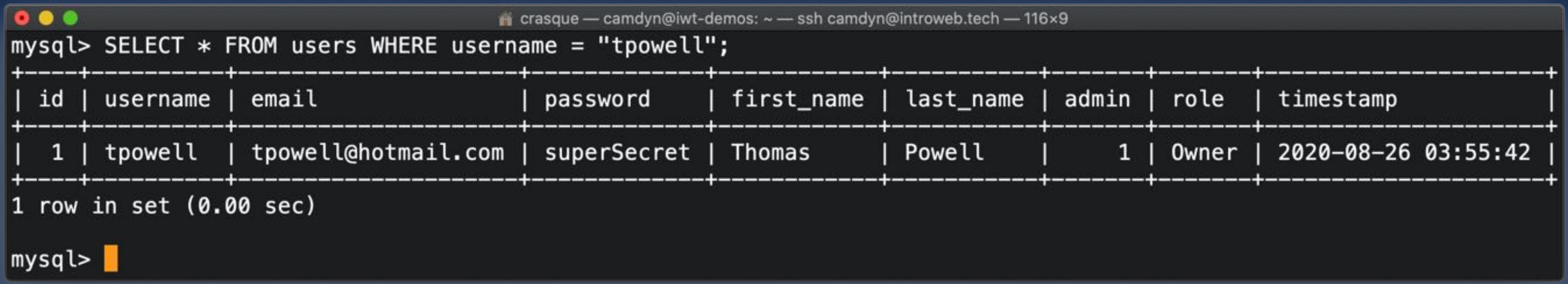
MongoDB


crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 76×14

```
> db.users.find()
{ "_id" : ObjectId("5f47f060ef6a5e2670b5a9a4"), "username" : "tpowell", "ema
il" : "tpowell@hotmail.com", "password" : "superSecret", "first_name" : "Tho
mas", "last_name" : "Powell", "admin" : true, "role" : "Owner", "timestamp"
: "10:41:00am Thu 08/27/20" }
{ "_id" : ObjectId("5f47f394ef6a5e2670b5a9a5"), "username" : "crasque", "ema
il" : "crasque@hotmail.com", "password" : "superSecret", "first_name" : "Cam
dyn", "last_name" : "Rasque", "admin" : false, "role" : "Dev", "timestamp" :
 "10:42:00am Thu 08/27/20" }
{ "_id" : ObjectId("5f47f394ef6a5e2670b5a9a6"), "username" : "jmartens", "em
ail" : "jmartens@hotmail.com", "password" : "superSecret", "first_name" : "J
osh", "last_name" : "Martens", "admin" : false, "role" : "Guest", "timestamp
" : "10:43:00am Thu 08/27/20" }
> 
```

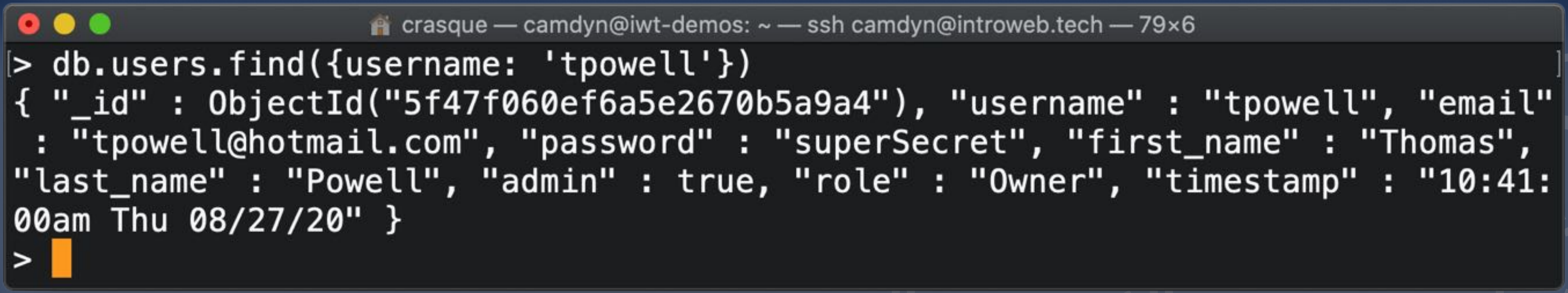
More Selecting Data

MySQL


crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 116×9

```
mysql> SELECT * FROM users WHERE username = "tpowell";
+----+----------+---------------------+-------------+------------+-----------+-------+-------+---------------------+
| id | username | email               | password    | first_name | last_name | admin | role  | timestamp           |
+----+----------+---------------------+-------------+------------+-----------+-------+-------+---------------------+
|  1 | tpowell  | tpowell@hotmail.com | superSecret | Thomas     | Powell    |     1 | Owner | 2020-08-26 03:55:42 |
+----+----------+---------------------+-------------+------------+-----------+-------+-------+---------------------+
1 row in set (0.00 sec)

mysql> 
```

MongoDB

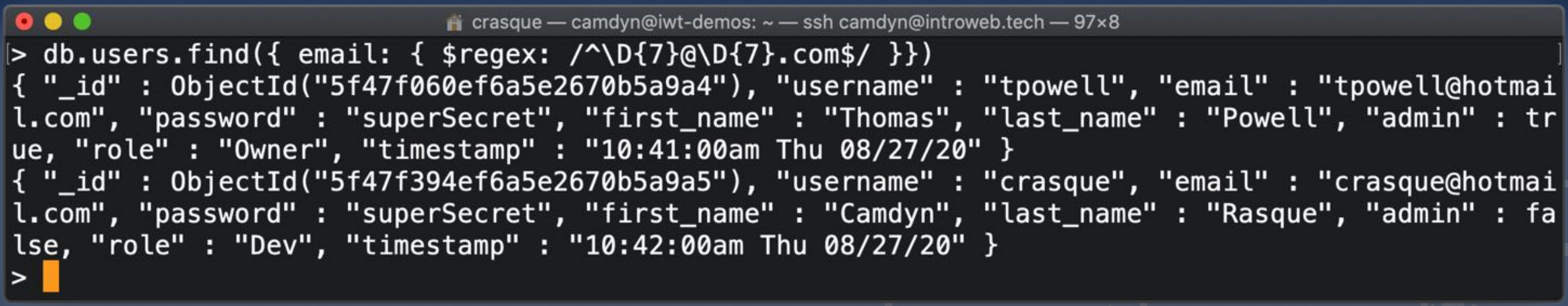

crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 79×6

```
> db.users.find({username: 'tpowell'})
{ "_id" : ObjectId("5f47f060ef6a5e2670b5a9a4"), "username" : "tpowell", "email"
 : "tpowell@hotmail.com", "password" : "superSecret", "first_name" : "Thomas",
"last_name" : "Powell", "admin" : true, "role" : "Owner", "timestamp" : "10:41:
00am Thu 08/27/20" }
> 
```

More Selecting Data

- MongoDB supports regex when querying, so pattern matching fields is as flexible as you need

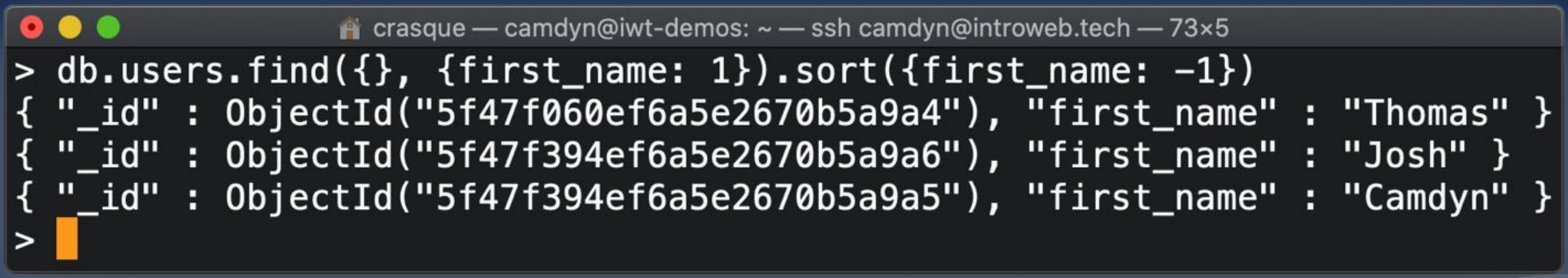
- This example is a pattern to find a string of exactly 7 non-digits followed by an @, followed by 7 more non-digits, ending in a .com


crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 97×8

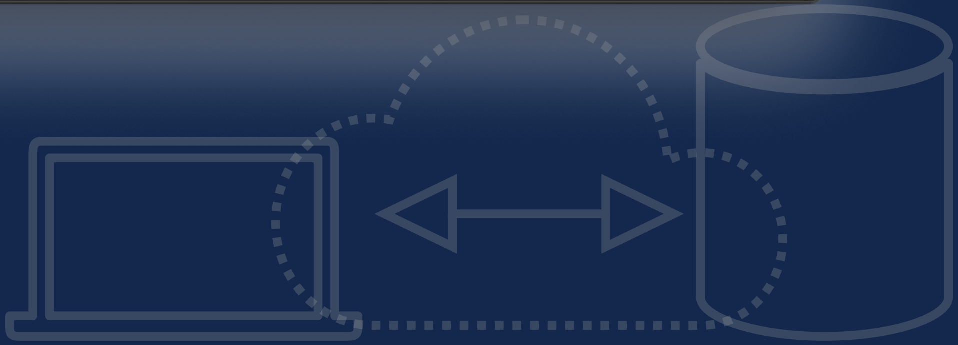
```
> db.users.find({ email: { $regex: /^\D{7}@\D{7}.com$/ }})
{ "_id" : ObjectId("5f47f060ef6a5e2670b5a9a4"), "username" : "tpowell", "email" : "tpowell@hotmai
l.com", "password" : "superSecret", "first_name" : "Thomas", "last_name" : "Powell", "admin" : tr
ue, "role" : "Owner", "timestamp" : "10:41:00am Thu 08/27/20" }
{ "_id" : ObjectId("5f47f394ef6a5e2670b5a9a5"), "username" : "crasque", "email" : "crasque@hotmai
l.com", "password" : "superSecret", "first_name" : "Camdyn", "last_name" : "Rasque", "admin" : fa
lse, "role" : "Dev", "timestamp" : "10:42:00am Thu 08/27/20" }
>
```

Sorting Data

- Add .sort() to a .find() call
  - Many different ways to sort, but to do a simple ascending/descending sort you specify the field name and then a 1/-1 respectively

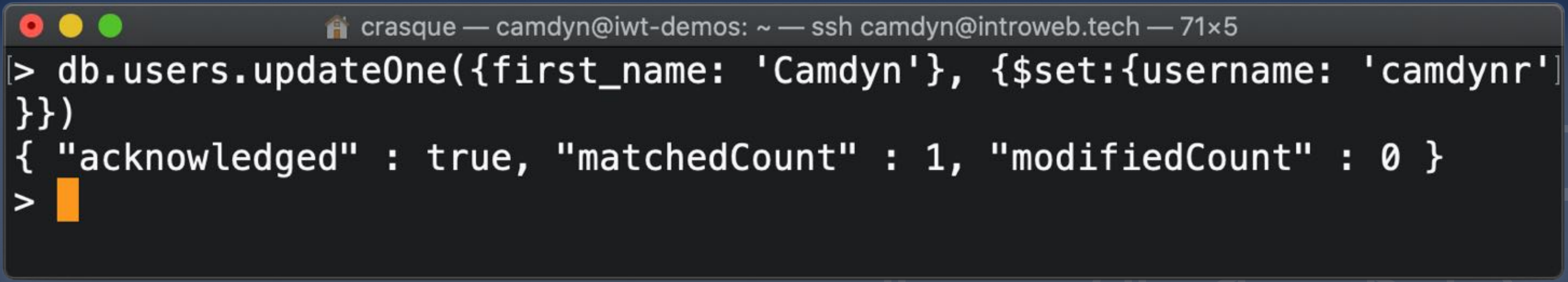

crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 73×5

```
> db.users.find({}, {first_name: 1}).sort({first_name: -1})
{ "_id" : ObjectId("5f47f060ef6a5e2670b5a9a4"), "first_name" : "Thomas" }
{ "_id" : ObjectId("5f47f394ef6a5e2670b5a9a6"), "first_name" : "Josh" }
{ "_id" : ObjectId("5f47f394ef6a5e2670b5a9a5"), "first_name" : "Camdyn" }
>
```



Updating Data

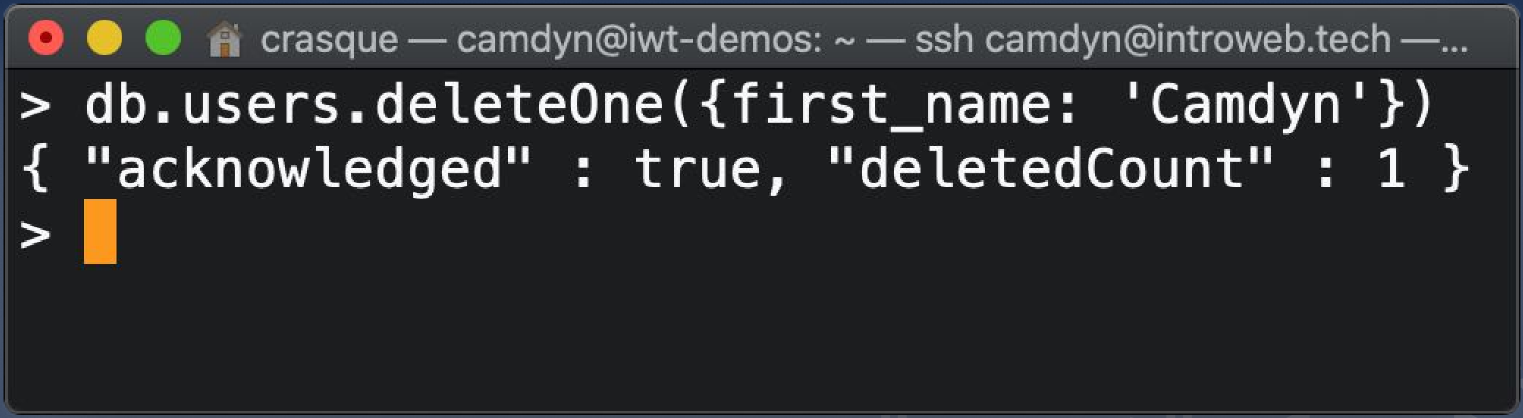
- The .updateOne() (and .updateMany() method respectively) take in two main parameters: A query of which documents to update, and an object with information you want to update the object with
  - There are a few ways to modify the object, but here we use `$set` to let mongo know that we want to replace the value of a field with this new value


crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech — 71×5

```
> db.users.updateOne({first_name: 'Camdyn'}, {$set:{username: 'camdynr'
}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
> 
```

Deleting Data

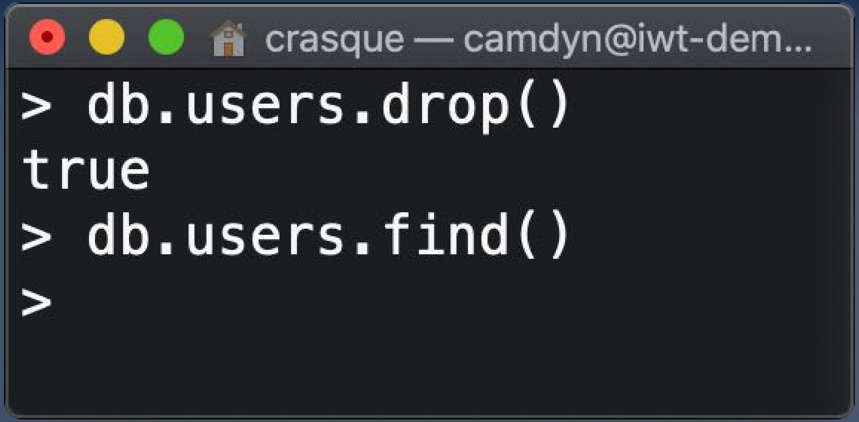
- The .deleteOne() method deletes a single document. If there are multiple that match the query, it will delete the first one it comes across.
- The .deleteMany() method, as the name implies, will delete all matches from its query


crasque — camdyn@iwt-demos: ~ — ssh camdyn@introweb.tech —...

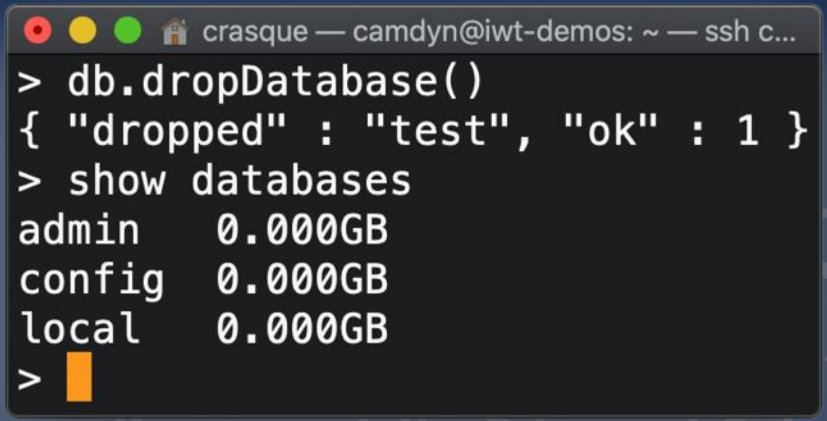
```
> db.users.deleteOne({first_name: 'Camdyn'})
{ "acknowledged" : true, "deletedCount" : 1 }
> 
```

Deleting Data – Big Changes

- Mongo retains the same *drop* keyword from SQL, so deleting a collection and its contents is as simple as db.collection.drop()
- To delete the entire database, db.dropDatabase() will drop whatever database you are currently using

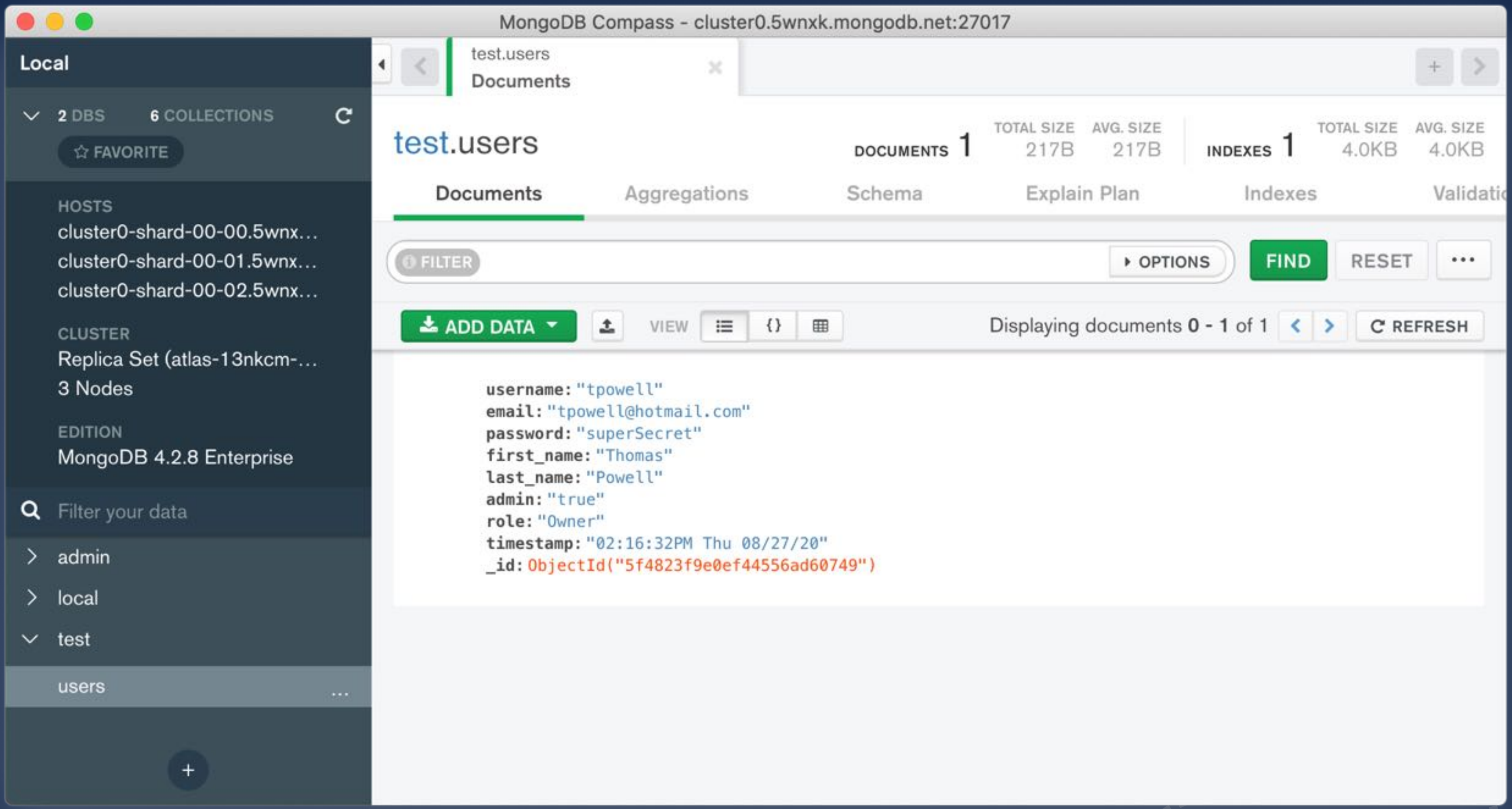


```
> db.users.drop()
true
> db.users.find()
>
```



```
> db.dropDatabase()
{ "dropped" : "test", "ok" : 1 }
> show databases
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

Easier Way - GUI Client

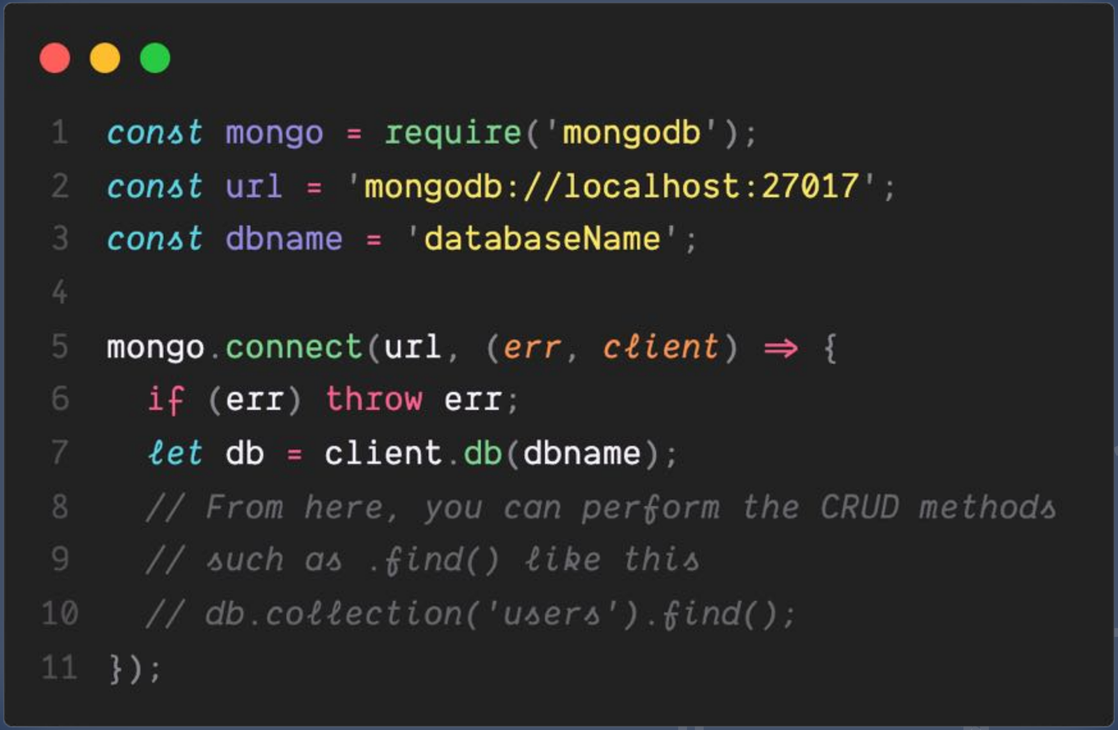


```
username: "tpowell"
email: "tpowell@hotmail.com"
password: "superSecret"
first_name: "Thomas"
last_name: "Powell"
admin: "true"
role: "Owner"
timestamp: "02:16:32PM Thu 08/27/20"
_id: ObjectId("5f4823f9e0ef44556ad60749")
```

https://www.mongodb.com/try/download/compass

Talking to MongoDB from NodeJS

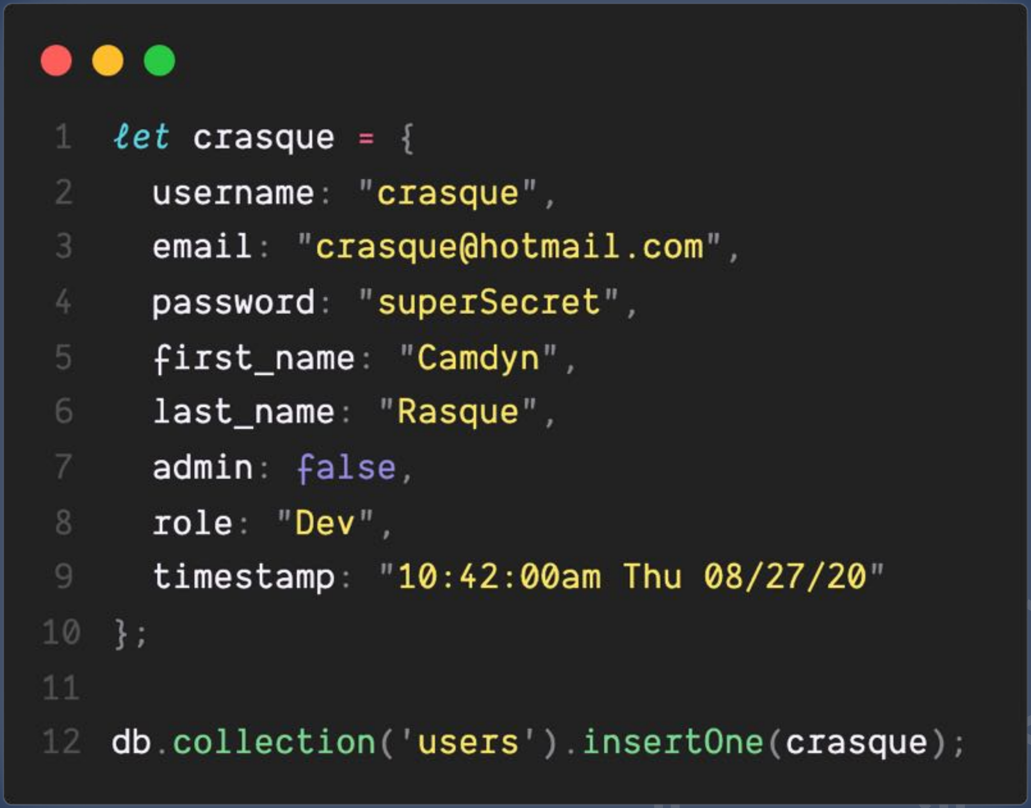
- First you need to include the mongodb package using require
- Then, grab the URL of your database. If it’s hosted on your local server, it will be on port 27017 unless specified otherwise.
    - Use mongo.connect to connect to mongo, and client.db() to connect to your db



```
const mongo = require('mongodb');
const url = 'mongodb://localhost:27017';
const dbname = 'databaseName';

mongo.connect(url, (err, client) => {
  if (err) throw err;
  let db = client.db(dbname);
  // From here, you can perform the CRUD methods
  // such as .find() like this
  // db.collection('users').find();
});
```

Talking to MongoDB from NodeJS

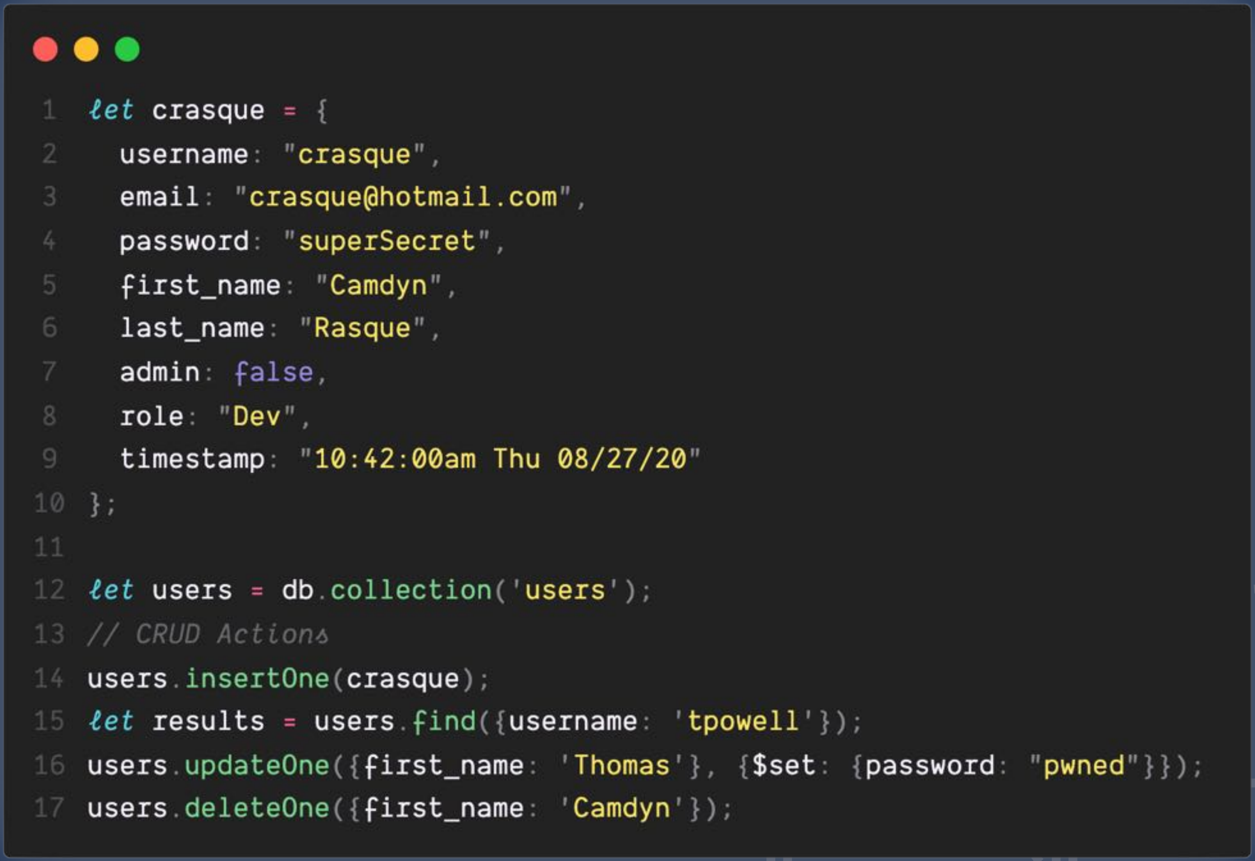
- Luckily, since the MongoDB shell uses JavaScript, talking to mongo from NodeJS is almost identical



```
let crasque = {
  username: "crasque",
  email: "crasque@hotmail.com",
  password: "superSecret",
  first_name: "Camdyn",
  last_name: "Rasque",
  admin: false,
  role: "Dev",
  timestamp: "10:42:00am Thu 08/27/20"
};

db.collection('users').insertOne(crasque);
```

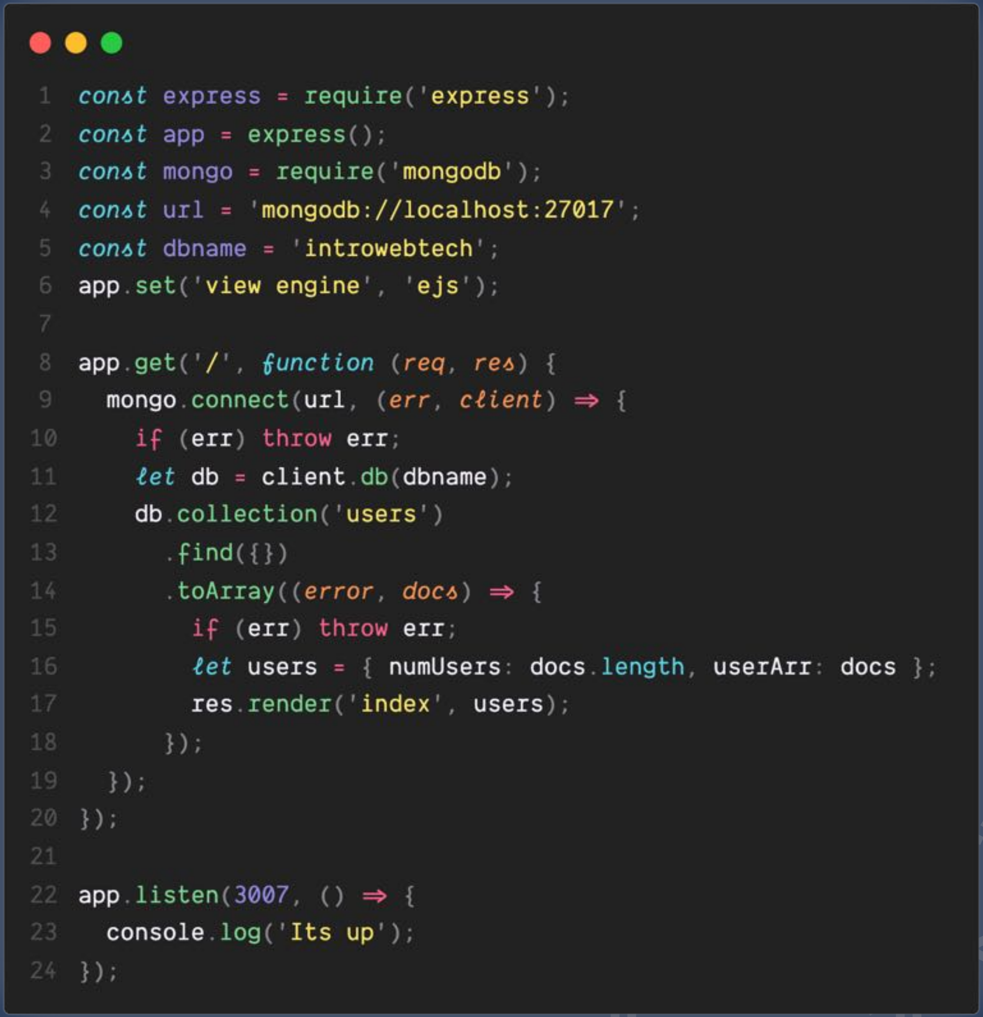
CRUD in MongoDB from NodeJS

- The rest of the CRUD actions, same as earlier



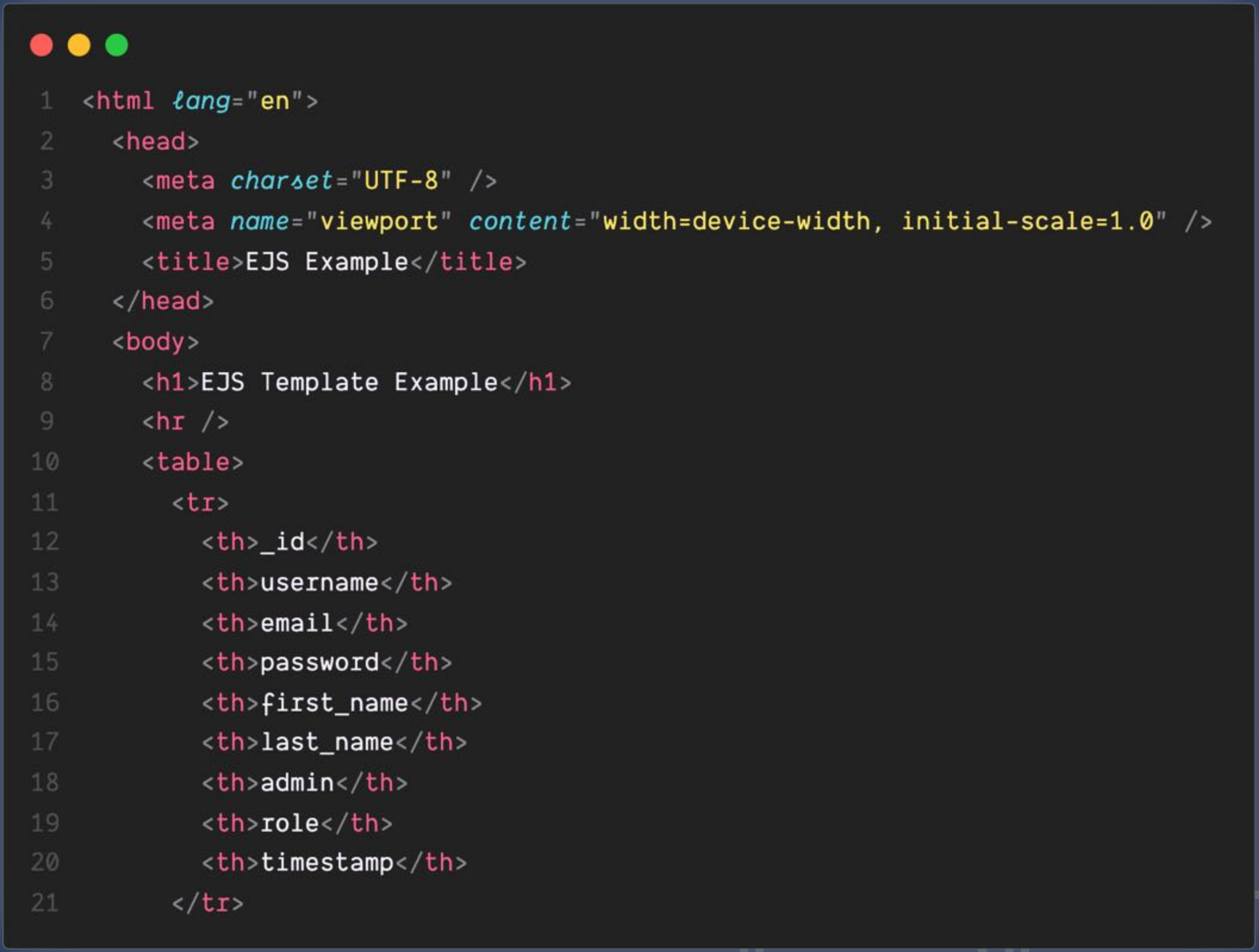
```javascript
let crasque = {
  username: "crasque",
  email: "crasque@hotmail.com",
  password: "superSecret",
  first_name: "Camdyn",
  last_name: "Rasque",
  admin: false,
  role: "Dev",
  timestamp: "10:42:00am Thu 08/27/20"
};

let users = db.collection('users');
// CRUD Actions
users.insertOne(crasque);
let results = users.find({username: 'tpowell'});
users.updateOne({first_name: 'Thomas'}, {$set: {password: "pwned"}});
users.deleteOne({first_name: 'Camdyn'});
```

EJS DB Example - Express Server



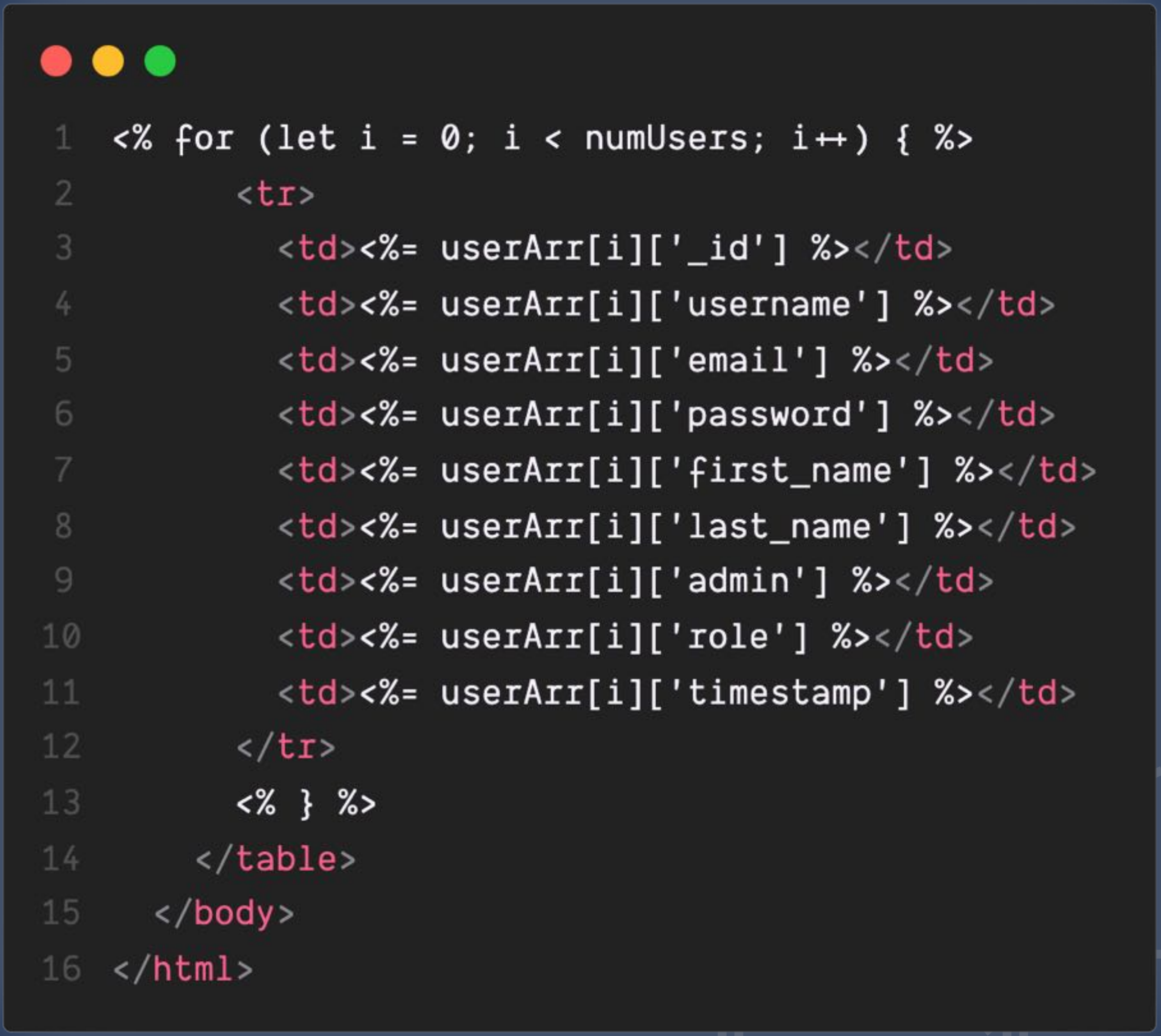
```javascript
const express = require('express');
const app = express();
const mongo = require('mongodb');
const url = 'mongodb://localhost:27017';
const dbname = 'introwebtech';
app.set('view engine', 'ejs');

app.get('/', function (req, res) {
  mongo.connect(url, (err, client) => {
    if (err) throw err;
    let db = client.db(dbname);
    db.collection('users')
      .find({})
      .toArray((error, docs) => {
        if (err) throw err;
        let users = { numUsers: docs.length, userArr: docs };
        res.render('index', users);
      });
  });
});

app.listen(3007, () => {
  console.log('Its up');
});
```

EJS DB Example - EJS Template Pt. I



```
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>EJS Example</title>
  </head>
  <body>
    <h1>EJS Template Example</h1>
    <hr />
    <table>
      <tr>
        <th>_id</th>
        <th>username</th>
        <th>email</th>
        <th>password</th>
        <th>first_name</th>
        <th>last_name</th>
        <th>admin</th>
        <th>role</th>
        <th>timestamp</th>
      </tr>
```

EJS DB Example - EJS Template Pt. II



```
<% for (let i = 0; i < numUsers; i++) { %>
      <tr>
        <td><%= userArr[i]['_id'] %></td>
        <td><%= userArr[i]['username'] %></td>
        <td><%= userArr[i]['email'] %></td>
        <td><%= userArr[i]['password'] %></td>
        <td><%= userArr[i]['first_name'] %></td>
        <td><%= userArr[i]['last_name'] %></td>
        <td><%= userArr[i]['admin'] %></td>
        <td><%= userArr[i]['role'] %></td>
        <td><%= userArr[i]['timestamp'] %></td>
      </tr>
      <% } %>
    </table>
  </body>
</html>
```

Summary

- NoSQL databases are performant and useful for more unstructured databases.
- NoSQL databases are not great with relation style queries and are less performant in that case compared to a properly designed SQL system
- Use of an example NoSQL like MongoDB shows that it naturally fits with languages like JavaScript and the use of objects avoiding needing a full fledged ORM
- The ease of NoSQL can have an evil side-effect of underplanning which can manifest in performance and maintenance issues. However, the ease also makes it useful for prototyping and there is no reason to consider SQL and NoSQL some sort of mutually exclusive choice or a side to pick as you may need both in a project or find both used at different times or for different needs