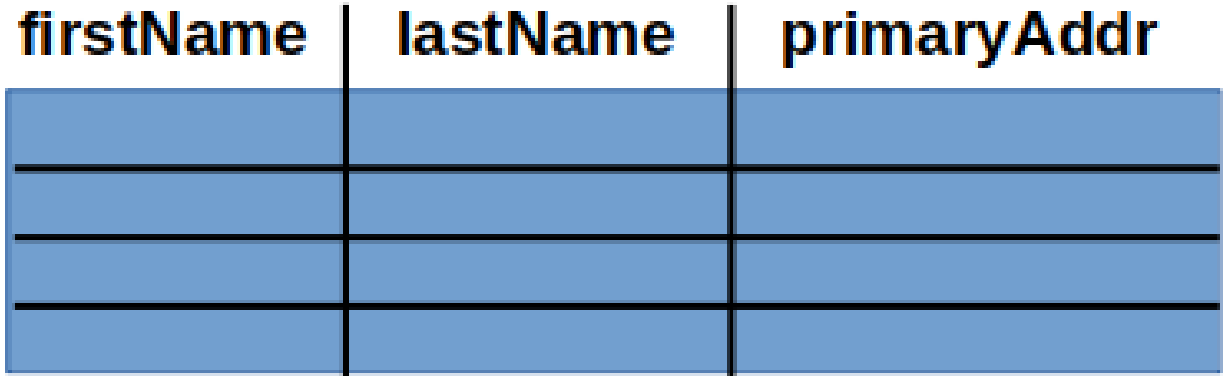
Introduction to Database Systems: CS312 MongoDB

Oliver BONHAM-CARTER

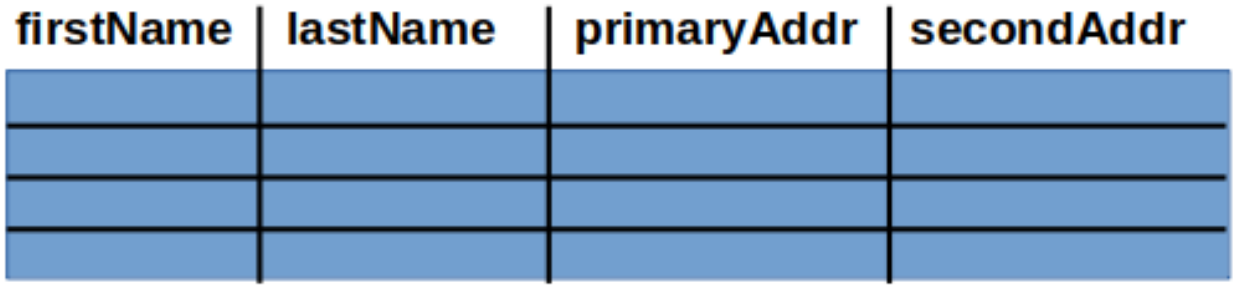
26 April 2022

The Problem with SQL



- Let's say that we have a (perfectly) working SQL table
- The schema has been designed and coded for *current data requirements*

Table Update (i)



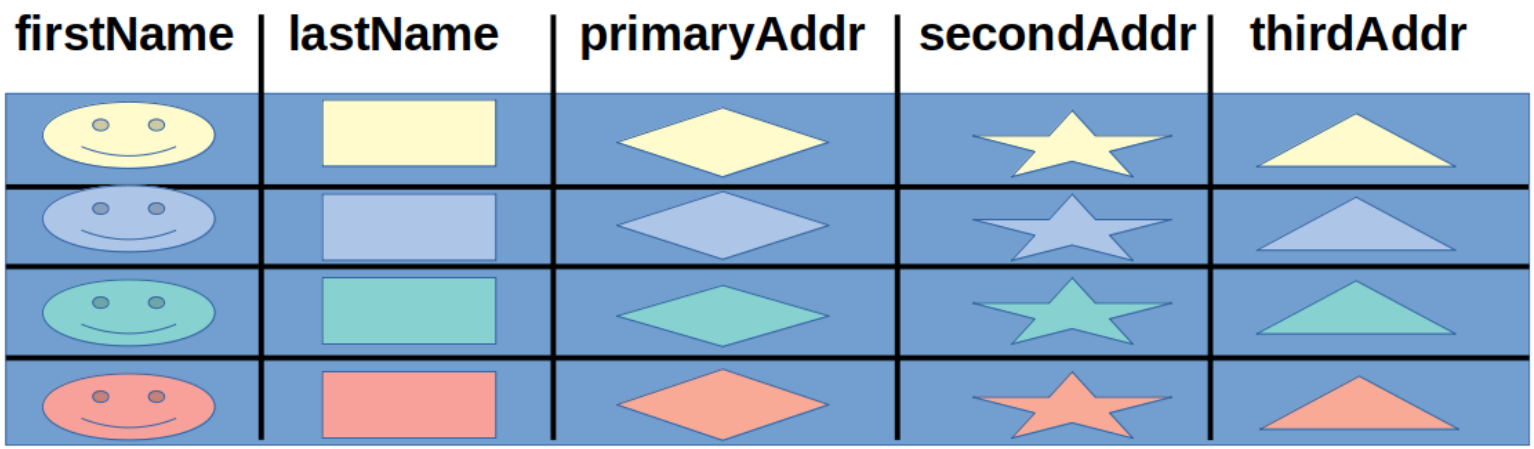
- The data we collect has changed.
- We need to update our schema for the *new data requirements*

Table Update (ii)

firstName	lastName	primaryAddr	secondAddr	thirdAddr

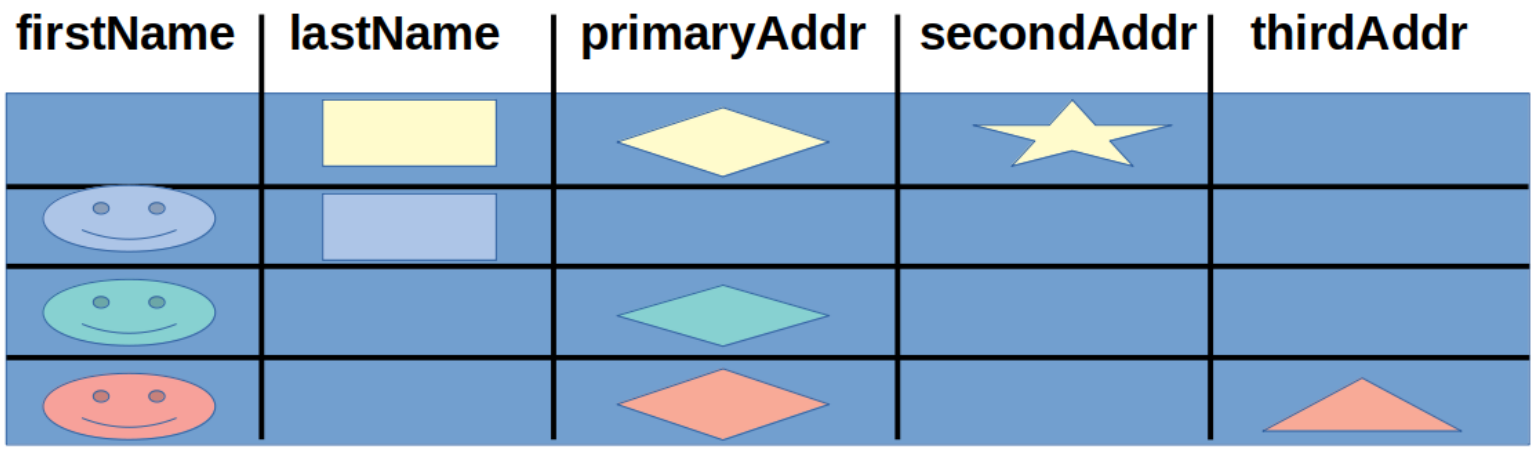
- Our needs have changed again and the SQL table must be updated.
- The schema is reprogrammed

Expectations



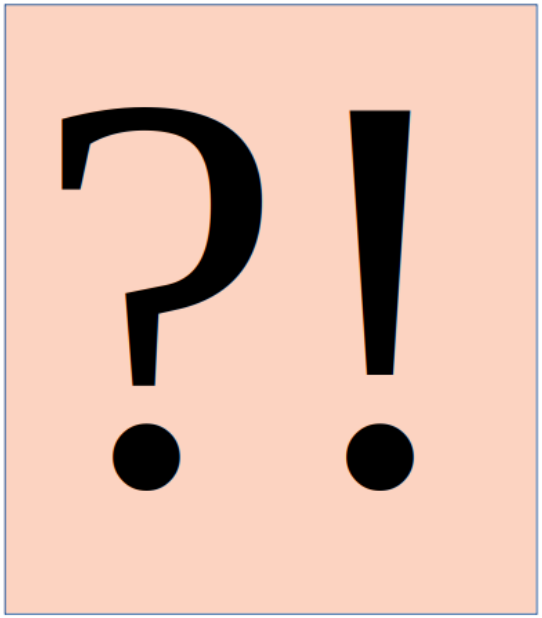
- We expect that the table will be full when in use
- Expectations are not always fulfilled...

In Reality, Much Data is Missing



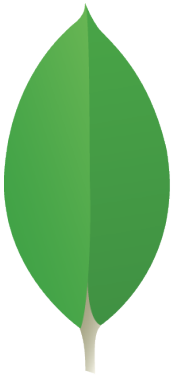
- But, in reality, much of the table is empty!
- The table can easily get huge and be hard to manage.

We Might Stop and Ask Ourselves...



- What can we do to stop having to redesign our database schema with our changing data?
- Is SQL the right type of database management system for our changing data requirements?

A NoSQL Database Management System



mongoDB®

- NoSQL: *Not Only SQL* database systems that support SQL-like query languages, but are used increasingly in big data applications and real-time web applications.
- The stored data is allowed to change

- `https://www.mongodb.com/`

Philosophy of MongoDB
Non-relational DB

- Document Identifiers (_id) will be created for each document, field name reserved by system
- Application tracks the schema and mapping
- Uses JSON, BSON (*B* for binary inputs)
- Written in C++
- Supports APIs (drivers) in many computer languages
  - JavaScript, Python, Golang, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

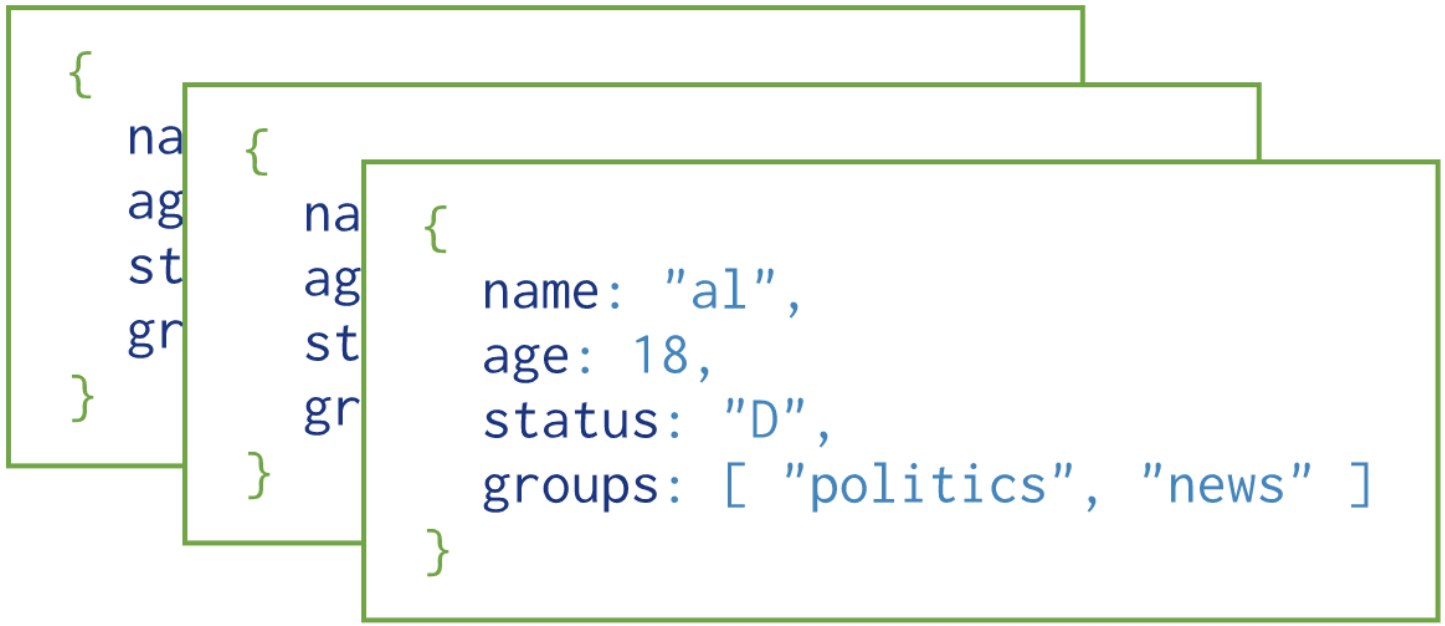
Database Language Guide

SQL systems versus NoSQL

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

- The terms are different but their meanings are similar
- *Schema*-less, collections (like tables) are populated by any data
- *Documents* are similar to the *tuples* of Sqlite3 programming

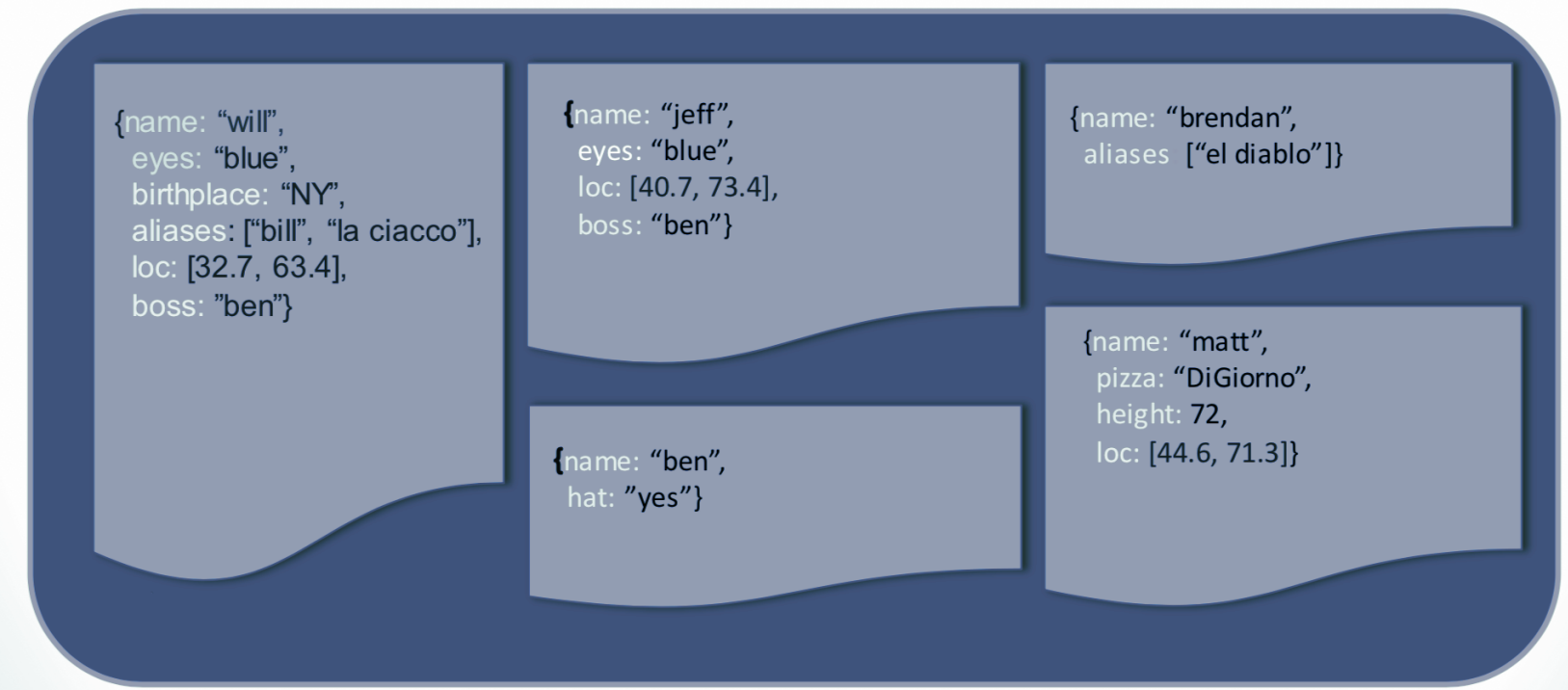
Schema Free



- No pre-defined data schema
  - Data may be entered in absence of a defined schema
- **Documents** (*rows*) of **collections** (*DB*'s) may have different types of data

Schema Free

Mostly similar documents



- Sometimes not all the data is available to create a document.
- The query *interprets* missing data as NULL entries

Styles of Storing Data

Relational Database

Student_Id	Student_Name	Age	College
1001	Chaitanya	30	Beginnersbook
1002	Steve	29	Beginnersbook
1003	Negan	28	Beginnersbook

MongoDB

```
{
 "_id": ObjectId("....."),
 "Student_Id": 1001,
 "Student_Name": "Chaitanya",
 "Age": 30,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1002,
 "Student_Name": "Steve",
 "Age": 29,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1003,
 "Student_Name": "Negan",
 "Age": 28,
 "College": "Beginnersbook"
}
```

JSON and MongoDB Code

- **Data is in name / value pairs**
- A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: `{ "name": "R2-D2 "}`
- **Data is separated by commas**
  - Example: `{ "name": "R2-D2", race : "Droid"}`
- **Curly braces hold objects**
  - Example: `{"name": "R2-D2", race : "Droid", affiliation: "rebels"}`
- **An array is stored in brackets** `[]`
  - Example `[ { "name": "R2-D2", race : "Droid", affiliation: "rebels"}, { "name": "Yoda", affiliation: "rebels"} ]`

CRUD Operations

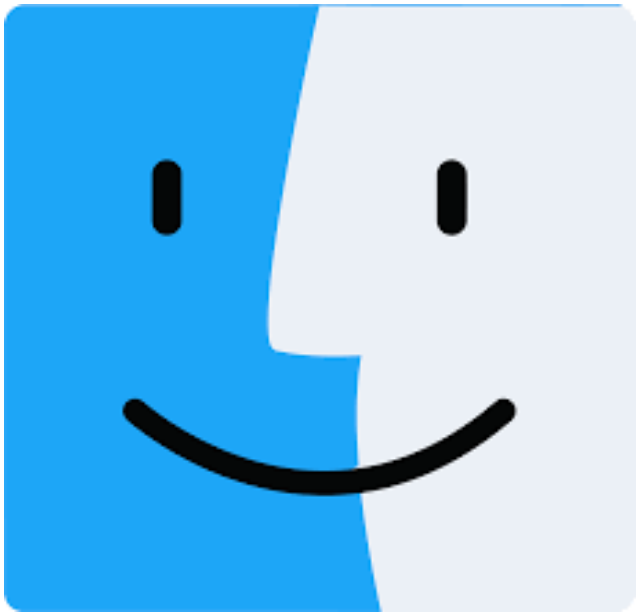
Create, Read, Update and Delete

We will see more of these commands later!

- **Db.collection** specifies the collection or the *table* in which to store the document (*tuple*)
- **Create**
  - `db.collection.insert()`
  - `db.collection.save()`
  - `db.collection.update()`
- **Read**
  - `db.collection.find()`
  - `db.collection.findOne()`
- **Update**
  - `db.collection.update()`
- **Delete**
  - `db.collection.remove()`

Setup Mongo in Docker Container

MacOS and Linux

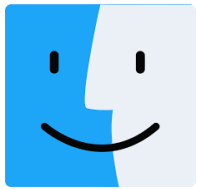


Create a Docker container from DockerHub: MacOS and Linux.

```
sudo docker pull mongo
```

Setup Mongo in Docker Container

MacOS and Linux, Terminal



Create a directory for data to persist.

```
mkdir -p ~/mongodata
```

Start a Docker container running the Mongo DB server

```
sudo docker run -it -v ~/mongodata:/data/db --name mongodb -d mongo
```

Check log to see that the server is operational

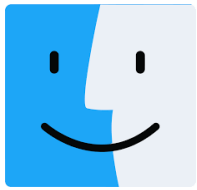
```
sudo docker logs mongodb
sudo docker ps
```

Run instance of MongoDB, goes into `root` of container.

```
sudo docker exec -it mongodb bash
```

Setup Mongo in Docker Container

MacOS and Linux, Terminal



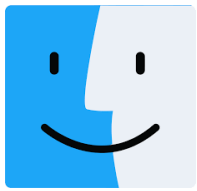
Start the MongoDB client

```
mongo
```

You are now able to run MongoDB commands here.

Setup Mongo in Docker Container

MacOS and Linux, Terminal



Leave the container

```
exit
```

Stop MongoDB container

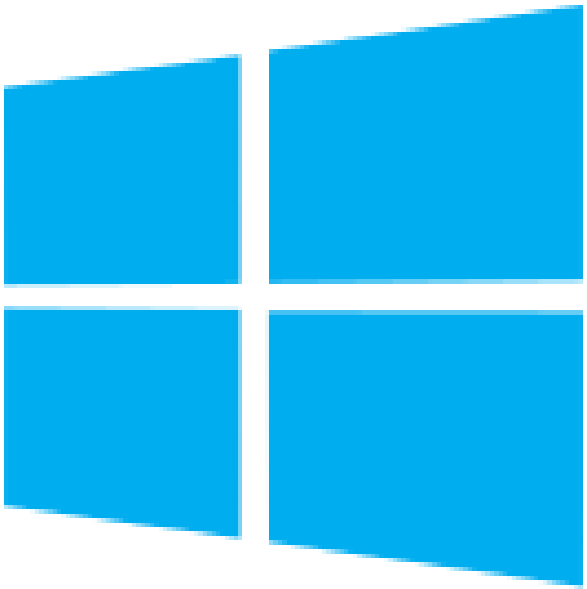
```
sudo docker stop mongodb
```

Removing all stopped containers, if necessary due to errors in launching container

```
sudo docker rm $(docker ps -a -q)
```

Setup Mongo in Docker Container

Windows, Powershell



Create a Docker container from DockerHub: Windows.

```
docker pull mongo
```

Setup Mongo in Docker Container

Windows, Powershell

Create a local directory for your data

```
mkdir c:\mongodata
```

Start a Docker container running the Mongo DB server

```
docker run -it -v c:\mongodata:/data/db --name mongodb -d mongo
```

Check log to see that the server is operational

```
docker logs mongodb
docker ps
```

Run instance of MongoDB, goes into root of container.

```
docker exec -it mongodb bash
```

Setup Mongo in Docker Container

Windows, Powershell

Start the MongoDB client

```
mongo
```

You are now able to run MongoDB commands here.

Setup Mongo in Docker Container

Windows, Powershell

Leave the container

```
exit
```

Stop Mongo container

```
docker stop mongodb
```

Removing all stopped containers, if necessary due to errors in launching container

```
docker rm $(docker ps -a -q)
```

Killing a Docker Container

If you have trouble running it, try relaunching the container

Find out the container's ID

```
docker ps
```

Your container's ID. This one is; `4e9d863407ac`

```
CONTAINER ID        IMAGE
4e9d863407ac        mongo
```

Stop MongoDB container

```
docker stop mongodb # Windows
sudo docker stop mongodb # MacOS and Linux
```

Remove MongoDB container

```
docker rm 4e9d863407ac # Windows
sudo docker rm 4e9d863407ac # MacOS and Linux
```

You should now be able to run your run and execute commands from above.

How to shut down a session

Drop the collection (called, *myCollection*): Destroy the data, remove collection

```
db.myCollection.drop()
```

Closing down

- Type `exit` in the client terminal

Simple Collections

Enter code into Mongo session of Docker container

Insert many documents into the *Furniture* collection

```
db.Furniture.drop()

db.Furniture.insert({chair:"wood"})
db.Furniture.insert({chair:"metal"})
db.Furniture.insert({chair:"plastic"})

db.Furniture.insert({table:"glass"})
db.Furniture.insert({table:"wood"})
db.Furniture.insert({table:"metal"})

db.Furniture.insert({lamp:"brass"})
db.Furniture.insert({lamp:"glass"})
db.Furniture.insert({lamp:"silver"})
```

```
db.Furniture.find({searchSpace},{showAttrib:1})
```

Simple Collections
Simple Example of Queries

Query all documents in the *Furniture* collection

```
db.Furniture.find({},{})
```

Query all *Lamp* types across all collections

```
// SELECT * FROM Furniture WHERE lamp == ``brass'';

db.Furniture.find({},{lamp:1})
db.Furniture.find({},{lamp:1,_id:0})
```

Query *Lamp* types from the *Furniture* collection

```
// SELECT lamp FROM Furniture WHERE lamp == ``brass'';

db.Furniture.find({lamp:"brass"})
db.Furniture.find({lamp:"glass"})
db.Furniture.find({lamp:"silver"})
// do not show object id's
db.Furniture.find({lamp:"silver"},{_id:0})
```

Insert many documents into the *Inventory* collection

Inserting

```
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

SELECT * FROM inventory

```
db.inventory.find( {},{} )
db.inventory.find( {},{} ).pretty()
```

SELECT item FROM inventory

```
db.inventory.find({},{"item":1}).pretty()
```

SELECT * FROM inventory WHERE item == “postcard”

```
db.inventory.find({"item":"postcard"},{})
db.inventory.find({"item":"postcard"},{}).pretty()
```

Queries from the *Inventory* collection

SELECT * FROM inventory WHERE status = "D"

```
db.inventory.find( { status: "D" } )
```

SELECT * FROM inventory WHERE status in ("A", "D")

```
db.inventory.find({status:{ $in: [ "A", "D" ]}}))
```

SELECT * FROM inventory WHERE status == "D")

```
db.inventory.find({ status: "D" },{})
```

Show me where the *size* = "h" and *size* = 10)

```
db.inventory.find( {"size.h":10} ).pretty()
```

See more on this at ...

https://www.mongodb.com/docs/manual/tutorial/query-documents/

Consider this...

- Can you go back to the above examples to query other fascinating information?
- Can you create and populate a new Mongo database?
- Can you write sophisticated queries in your database?