**Ch. 24, 25, 26, 27 – Databases, SQL and MongoDB, Mongoose**

Databases can be comprised of either:

SQL – structured query language. Old and reliable. More structured (requires a schema). “relational DB” (can have multiple tables in relation with each other to easier store relevant data and reduce repetition – customer table, product table, orders table). Scales vertically (not good).

Organizes data in tables like excel. Inserts null in empty data fields.

NoSQL – Not only structured query language. Includes SQL and another language. Tends to be newer. More flexible and not bound to a particular structure. “nonrelational DB” (better for 1 user: many posts, style). This type of DB allows for greater scalability as it uses a distributed system. Scales horizontally.

Data stored as JSON objects.

Top 4:

SQL: SQL MySQL, PostgreSQL

NoSQL: MongoDB and redis

**SQL:**

Main functions are (CRUD):

CREATE

CREATE TABLE table\_name (

column1 datatype,

column2 datatype… )

Creates a table with columns of specified datatypes.

INSERT INTO table\_name(column1, column2…)

VALUES(value1, value2…)

This inserts values into the table specified.

You can avoid naming which columns you want to insert data into by providing values for every column in the table

READ

SELECT column FROM table\_name WHERE condition;

This shows us the column of the table specified with the selected condition

SELECT \* selects all columns of the given table.

WHERE statement is optional

UPDATE

UPDATE table\_name

SET column1 = value1, column2 = value2…

WHERE condition;

This updates the values of the selected column of the selected table with the restricting condition. You can update multiple values at once.

ALTER TABLE table\_name

ADD column3 datatype;

Adds a new column column3 of the specified datatype

DROP COLUMN column2;

Deletes the specified column

ALTER COLUMN column1 newDatatype;

Changes the datatype of the specified column to newDatatype

DELETE

DELETE FROM table\_name

WHERE condition;

Deletes anything in the specified table with the specified condition

**How to link tables together:**

When you create a new table, you can add:

FOREIGN KEY (keyName) REFERENCES table\_name(primaryKeyName)

This links the newly created table’s keyName to the previous table\_name’s primary key primaryKeyName

If you want to create a new table with different columsn of different tables:

SELECT orders.order\_number, products.product, products.price, products.stock

FROM orders

INNER JOIN products ON orders.product\_id = products.id;

This combines different columns from different tables into one by combing the table orders with the table products. The values are identified by matching orders.product\_id with products.id.

**MongoDB:**

**How to create (databases, collections and documents):**

use databaseName

Creates a database with name databaseName, otherwise, switches to the DB specified.

db

Shows you which database you are currently in

show dbs

Lists all of the databases present

db.collectionName.insertOne({document})

collectionName is the name of your collection. If it does not exist, it is made.

Document comprises of fields and values. field: value, field2: value2, …

This is similar to key: value pairs of JS objects.

The first field is the \_id key which acts as a primary key for each document. It must be unique.

db.collection.insertMany({documents})

show collections

shows all of the collections in the given DB

**How to read:**

db.collectionName.find(query, projection)

searches the specified collection of the current db with the specified query and projection. Both arguments are optional.

db.collectionName.find(

{ age: { $gt: 18 } },

{ name: 1, address: 0 }

).limit(5)

Query restricts your search. Projection specifies which fields you would like displayed. 1 for projection means to display, 0 means do not display. For the example, this searches for all age fields with values over 18. It will display the names of those documents but not the addresses.

**How to update:**

db.collectionName.updateOne({\_id: 1}, {$set: {fieldName: value1}})

this updates \_id: 1 of collectionName to add fieldname: value1.

**How to delete:**

db.collectionName.deleteOne({filter: value})

This deletes the specified documents of the collection specified.

db.dropDatabase()

This deletes the current DB and its contents.

**How to establish relationships in Mongo:**

You can embed documents in other documents in an array for example

db.collectionName.insertOne({

\_id: 1,

Name: “Pencil”,

Price: 0.8,

Stock: 12,

Reviews: [{name: “Sally”, rating: 5, review: “best pencil”}, {name: “John”, rating: 3, review: “its alright”}]

})

A different way to create relationships is to have 2 separate documents in a collections (products), then, in a different collection (orders), reference the 2 documents you want to connect.

**How to use MongoDB with Node:**

Use mongoose…

**How to use Mongoose with Node:**

**How to create:**

const mongoose = require("mongoose");

//requires the Mongoose NPM

mongoose.connect("mongodb://localhost:27017/fruitsDB", { useNewUrlParser: true, useUnifiedTopology: true });

//Connects to the MongoDB

const fruitSchema = new mongoose.Schema({

name: String,

rating: Number,

review: String });

//creates schema for your collection with the specified fields and value types

//A document cannot include a field which is not included in the classification’s schema.

const Fruit = mongoose.model("Fruit", fruitSchema);

//creates your collection. Param1 = Name of collection (usually in singular tense).

//Mongoose will lowercase your collection name (using lodash) and pluralize it.

//Param2 = Which schema to use.

const fruit1 = new Fruit({

name: "John",

review: "lets see if this works",

rating: 640 });

//Creating a document of the specified collection

person.save();

//inserts your fruit document in your Fruit collection of your fruitsDB

Fruit.insertMany([fruit1, fruit2, fruit3], function(err){

if (err){ console.log(err); } else{ console.log("successfully saved all fruits to fruitsDB"); } });

//inserts your fruit documents in your array (param1) to the specified collection and DB.

//param2 is a callback function which can confirm successful or unsuccesful exection of code.

**How to read:**

Fruit.find(function(err, fruits) {

if (err) { console.log(err); } else { fruits.forEach(function(fruit) { console.log(fruit.name); })

mongoose.connection.close(); } });

//searches your database and prints off the names field your documents

//closes your connection to mongoose when it is done console logging the values.

**How to validate data:**

const fruitSchema = new mongoose.Schema({

name: { type: String,

required: [true, "whats the name??!!"] },

rating: { type: Number,

min: 1,

max: 10 },

review: String });

//This schema states a name is required and returns a custom error message when one is not provided. It also limits the rating from between 1 – 10, both inclusive.

**How to update:**

Fruit.updateOne({ \_id: "5f2c335482f60a28047141e4" }, { rating: 6, review: "I suck" },

function(err) { if (err) { console.log(err); } else { console.log("updated successfully"); } });

//updates the specified value of the document

//param1 = the search criteria. param2 = what values you want to update, param3 = callback function to handle errors.

**How to delete:**

Fruit.deleteOne({name: "Apple"},

function(err) { if (err) { console.log(err); } else { console.log("deleted apple successfully"); } });

//deletes the selected document

//param1 = search criteria. param2 = callback function to handle errors.

//you can substitute our deleteOne() for deleteMany(). It uses the same parameters.

**How to establish relationships between documents:**

const personSchema = new mongoose.Schema({ name: String,

age: Number,

favoriteFruit: fruitSchema })

//relationships. You can embed a document in the field of another document.

//If you save both documents simultaneously, the ID of the fruit will match the id of the Fruit in the person’s document.