Databases

CHEAT SHEET

SQL VS NO SQL

SQL CRUD

SQL – Create
SQL – Update
SQL – Read
SQL – Delete

SQL Primary & Foreign key
SQL Joins

Mongo DB CRUD

MongoDB – Create
MongoDB – Read
MongoDB – Update
MongoDB – Delete

MongoDB - Relations

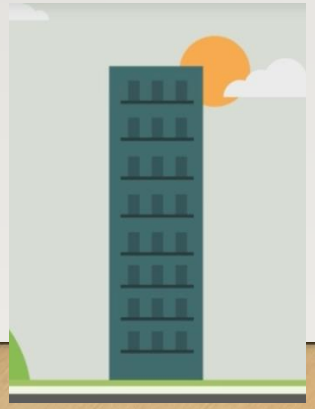
Mongoose

Connecting node app to MongoDB
Performing CRUD
Data validation
Implementing Relations

SQL VS NO SQL

The difference between 2 main types of databases explained …

SQL (Structured Query Language)	NO SQL (Not only Structured Query Language)
MySQL, PostgreSQL	MongoDB, Redis, Neo4j
○ More mature & have been used for a long time since the late 1960's	○ Shiny & new & have been introduced in the early 21st century
○ Data stored in tables & follows table structure	○ Data stored in json objects & follows document structure
○ Inflexible to changing requirements as it follows a pre defined structure (Schema)	○ Flexible to changing requirements
○ Great with relationships between data (Relational)	○ Not great with complex relationships between data (Non - relational)
○ They scale vertically	○ They scale horizontally & data can be distributed among computers



SQL VS NO SQL

The difference between 2 main types of databases explained …

SQL

- Follows table Structure
- Inflexible to changing requirements

Customers			
First Name	**Last Name**	**Address**	**Email**
John	Doe	32 Cherry Blvd	Null
Angela	Yu	12 Sunset Drive	angela@gmail.com
Jack	Bauer	Null	Null

NO SQL

- Follows Document structure
- Flexible to changing requirements

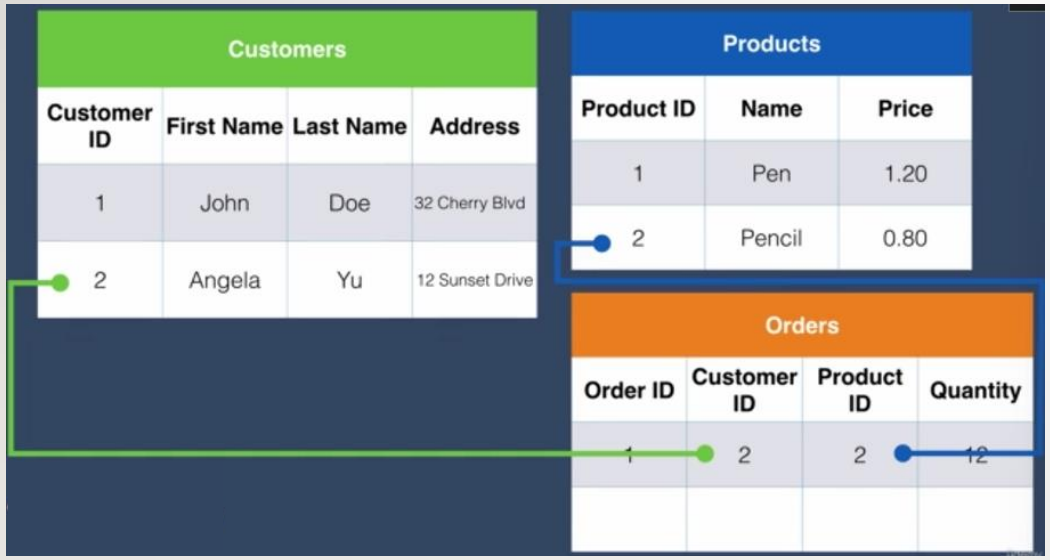
```
{
 first_name: “John”,
 last_name: “Doe”,
 address: “32 Cherry Blvd”
}
{
 first_name: “Angela”,
 last_name: “Yu”,
 address: “12 Sunset Drive”,
 email: “angela@gmail.com
}
{
 first_name: “Jack”,
 last_name: “Bauer”
}
```

SQL VS NO SQL

The difference between 2 main types of databases explained …

SQL

- Great with relationships



NO SQL

- Not Great with complex relationship's

```
{
   order_id: order_01,
   customer: {
     first_name: "Angela",
     last_name: "Yu",
     address: "12 Sunset Drive"
   },
   products: {
      product_name: "Pencil",
      product_price: 0.80
   },
   order_quantity: 12
}
```

Store everything in one object

```
{
   order_id: order_01,
   customer: person_01,
   products: product_01,
   order_quantity: 12
}

{
   person_id: person_01,
   first_name: "Angela",
   last_name: "Yu",
   address: "12 Sunset Drive"
}

{
   product_id: product_01
   product_name: "Pencil",
   product_price: 0.80
}
```

Store in multiple documents

SQL – Database operations

Create a Database :- create database db_name	Create database imdb;
Show all created Databases :-	show databases;
Use a particular database :- Use db_name	Use imdb;
Dump data from another SQL into DB :- source file_path_of_SQL_file	source D:\DBMS\imdb.sql;
Delete particular database :- Drop database db_name	Drop database imdb;

SQL CRUD - Create

Create table :- create table table_name (col_1 datatype, col_2 datatype, ... col_n datatype,);	Create table emp (eid int, name varchar(20), address char(25), salary decimal(18, 2));
Data is compulsory :- col_name datatype NOT NULL	Create table student (sid int NOT NULL);
Data should be Unique for every row :- col_name datatype UNIQUE	Create table student (username varchar(20) UNIQUE)
Data should satisfy user defined condition :- col_name datatype, check (col in ('val_1', 'val_2'))	Create table customer (gender char(1), check (gender in ('m','f')));
Add default value, if data not specified :-	Create table customer

SQL CRUD - Read

Display table info :- desc table_name;	desc products; or describe actors;
Select all columns/display full table :- select * from table_name;	select * from products;
Select specific columns :- Select col_name from table_name;	select name, price from products;
Select unique records :- Select DISTINCT col_name from table_name;	select DISTINCT name from products; **Q.) Select rows whose combination of first_name, last_name is unique** select distinct first_name, last_name from directors;
Select records based on specific condition :- select * from table_name where condition;	select * from products where id=1;
Q.) Find all the orders issued against the salesman who may works for customer whose id is 3007	Select * from orders where customer_id = 3007;
Q.) Display the commission of all the	Select commission from salesman

SQL – Data Query Language

LIMIT :- Limits no. of rows to be displayed select * from table_name limit limit_no.	select * From departments limit 3;
OFFSET :- Specifies no. of rows to ignore from start select * from table_name offset offset_no.;	**Q.) Display the 20 rows after the initial 20 rows** select name, rankscore from movies limit 20 offset 20;
ORDER BY :- Display records in sorted order **// Sort in ascending order** select * from table_name order by col_name; **// Sort in descending order** select * from table order by col_name desc;	**Q.) Show movies in descending order by year (Get recent movies)** select name, year from movies order by year desc; **Q.) Show movies in ascending order by year (Get old movies)** select name, year from movies order by year; **Q.) Order alphabetically (A→Z)** select first_name from directors order by first_name; **Q.) Order alphabetically (Z→A)** select first_name from directors order by first_name desc;

SQL - DQL (Comparison Operators)

Operator - (equal to =) **Note :-** equal operator does not work on NULL, for that we need to use the IS keyword	**Q.) Find movies whose genre is comedy** select * from movies_genres where genre = 'Comedy'; **Q.) Find movies whose rankscore is NULL** select name, year, rankscore from movies where rankscore IS NULL; **Q.) Find movies whose rankscore is Not NULL** select name, year, rankscore from movies where rankscore IS NOT NULL;
Operator - (not equal to <>)	**Q.) Find all faculty not teaching 'DT'** Select * from Faculty Where Subject <> 'DT';
Operator - (Greater than >)	**Q.) Find all faculty teaching more than 10 hours** Select * from Faculty Where Hours > 10;

SQL - DQL (Set Operators)

Set - (Union) Query_1 Union Query_2	Select * from stud **Union** Select * from stud_extern;
Set - (Intersect) Query_1 Intersect Query_2	Select * from stud **Intersect** Select * from stud_extern;
Set - (Except) Query_1 Except Query_2	Select * from stud **Except** Select * from stud_extern;

SQL – DQL (Logical Operators)

Logical – (AND)	**Q.) Find all faculty not teaching 'DT' and taught more than 10 hours** Select * from Faculty Where Subject <> 'DT' **AND** Hours > 10; **Q.) Find all movies whose rankscore is greater than 9 & released after year 2000** select name, year, rankscore from movies where rankscore>9 **AND** year>2000;
Logical - (OR)	**Q.) Find all faculty not teaching 'DT' or taught more than 10 hours** Select * from Faculty Where Subject <> 'DT' **OR** Hours > 10 ;
Logical - (NOT)	**Q.) Find all faculty not teaching more than 10 hours** Select * from Faculty Where **NOT** Hours > 10 ;
Logical – (BETWEEN) **Note :-** lowValue <= highvalue else empty result set will be returned Eg. Hours BETWEEN 20 AND 10 will return a empty set;	**Q.) Find all faculty teaching between 10 to 20 hours** Select * from Faculty Where Hours **BETWEEN** 10 AND 20; **Q.) Find all faculty not teaching between 10 to 20 hours**

SQL – DQL (Pattern matching using LIKE)

Logical - (Like) (Pattern matching) % → String of 0 or more characters _ → Single character	Select col_name From Table_name Where col_name Like {Pattern};
Q.) Find all faculties whose name starts with 'A'	Select * From Faculty Where Faculty_name Like 'A%';
Q.) Find all faculties whose name starts with 'Om'	Select * From Faculty Where Faculty_name Like 'Om%';
Q.) Find all faculties whose name ends with 'h'	Select * From Faculty Where Faculty_name Like '%h';
Q.) Find all faculties whose name contains letter 'a'	Select * From Faculty Where Faculty_name Like '%a%';
Q.) Find all faculties whose names second letter is 'a'	Select * From Faculty Where Faculty_name Like '_a%';
Q.) Find all faculties whose names second last letter is 's'	Select * From Faculty Where Faculty_name Like '%s_';
Q.) Find all faculties whose name do not contains 'a'	Select * From Faculty Where Faculty_name NOT Like '%a%';
Q.) Find all faculties whose names second letter is 'a', Contains letter 'e' and name ending with 'h'	Select * From Faculty Where Faculty_name Like '_a%e%h';
Q.) Find students who have 96% percentage	Select * From students

SQL – DQL (Aggregate functions)

SUM()	**Q.) Find the sum of total fare from ticket_header** select sum(fare) from ticket_header;
AVG()	**Q.) Give the average of total fare from ticket_header** select avg(fare) from ticket_header;
MAX()	**Q.) Find out the highest fare from ticket_details** select max(fare) from ticket_details;
MIN()	**Q.) Give the min distance from route_header** select min(dist) from route_header;
COUNT()	**Q.) Give the total collection of fare from ticket_details** select count(fare) from ticket details;
Q.) Find count of no. of movies whose year is greater than 2000	select count(*) from movies Where year>2000;
Q). Give the total no of people who have travelled more than 36hrs. Group by ticket no	select count(adult), count(child) From ticket_header Where time>36 Group by tikno;

SQL – Subqueries, nested & inner queries

First inner query is executed & then the outer query is executed using the output values in inner query

Nested queries :- First inner query is executed & then the outer query is executed using the output values in inner query Possible Operators :- in, not in, exists, not exists, any, all, comparison operators 1. IN checks where the records belongs to the set & returns the result accordingly 2. EXISTS returns true if the subquery returns one or more records or NULL 2. ANY operator returns true if any of the subquery values meet the condition 3. ALL Operator returns true if all of the subquery values meet the condition	// Syntax :- SELECT col_name From table_name WHERE col_name OPERATOR(SELECT col_name From table_name WHERE = (condition or subquery))
Q.) List all actors present in the movie Schindler's List actors(id, first_name, last_name, gender) roles(actor_id, movie_id, role) movies(id, name, year, rankscore)	select first_name, last_name from actors where id IN (select actor_id from roles where movie_id IN(select id from movies where name="Schindler's List"));
Q.) List all movies whose rankscore is same as	select * from movies

SQL CRUD - Update

Insert values/rows into table :- insert into table_name values(col_1_data, col_2_data, .. col_n_data);	Insert into products values (1, "PEN", 1.20); Insert into products values (1, “PEN”, 1.30), (2, “GLUE”, 2.00), (3, “CARD”, 1.10) Insert into products (id, name) values (4, "PENCIL");
Copying rows from other table :-	Insert into products select * from products1 where name in (“eraser”, “ruler”)
Update data values :- update table_name set col_name= data where condition;	update products set price = 0.80 where id=2;
Add/modify/drop columns into table :- Alter table table_name add/modify col_name datatype; Alter table table_name	Alter table products add stock int; Alter table products modify name varchar(50);

SQL CRUD - Delete

Delete specific row :- Delete from table_name where condition;	Delete from products where name = "pencil";
Delete only all table records :- Truncate table table_name	Truncate table products;
Delete full table & it's records :- Drop table table_name;	Drop table emp;

SQL – Joins

Join 2 tables :- Select * From T1 Join T2; Or Select * From T1 join T2 using (c1);	**Q.) Join order & products table based on primary key** SELECT orders.order_number, products.name, products.price FROM orders INNER JOIN products on orders.product_id = products.id; **Q.) For each movie present in movie table display name & genre present in movies_genres table** SELECT m.name, g.genre from movies m JOIN movies_genres g on m.id = g.movie_id; * Note m, g are table aliases
Join 3 tables :-	**Q.)Join customer, order & products table based on primary key** SELECT customers.id, customers.first_name, customers.last_name, orders.order_number, products.name, products.price FROM customers JOIN orders ON customers.id=orders.customer_id JOIN products ON products.id=orders.product_id
Left Outer join & Right Outer join :-	**Q.) Left outer join movie & movies_genres table** SELECT m.name, g.genre from movies m

SQL – Data control

GRANT :- Granting privileges to user's by allowing specified users to perform specified tasks	Create user 'jeff'@'localhost' identified by 'password'; grant ALL on db1 to 'jeff'@'localhost'; grant SELECT on db1 to 'jeff'@'localhost';
Revoke:- Revoking privileges from user's by canceling previously granted permissions	revoke DROP on db1 from 'jeff'@'localhost';

Mongo DB CRUD - Create

Create collection :- db.createCollection();	**// Create products collection** db.createCollection("products");
Insert records into collection :- db.collection.insertOne() db.collection.insertMany()	db.products.insertOne({ _id: 3, name: "Rubber", price: 1.50, reviews: [{ authorName: "Sally" rating: 5, review: "Awesome rubber!" }] }) db.products.insertMany([{ _id: 1, name: "pen", price: 1.20 }, { _id: 2, name: "pencil", price: 0.80 }])

Mongo DB CRUD - Read

Display collection/specific records :-

db.collection.find(query, projection)

- **Projection -** To specify which fields we want to see

```
// Display all records in pretty print
db.products.find().pretty()

// Display 1st record from collection
db.products.findOne()

// Display pencil products
db.products.find({name: "Pencil"})

// Display products with id 1 and show only name field
db.products.find({_id: 1}, {name:1})

// Display products having price > 1
db.products.find({price: {$gt: 1}})

// Display products having 0 < price < 1
db.products.find({price: {$gt: 0, $lt: 1}})

// Display products having price atleast 1
db.products.find({price: {"$not: {$lt: 1}}})

// Sort products by increasing price
db.products.find().sort({price:1})

// Sort products by decreasing price
db.products.find().sort({price:-1})
```

Mongo DB CRUD - Update

Update records in collection :- db.collection.updateOne()	**// Add stock field for id 1** db.products.updateOne({_id:1}, {$set: {stock: 32}})

Mongo DB CRUD – Delete

Delete records in collection :- db.collection.deleteOne()	**// Delete id 1 element** db.products.deleteOne({_id: 2})

Mongo DB – Relations

Implementing relationships in MongoDB :-

```
// Products collection
{
    _id: 1,
    name: "pen",
    price: 1.20
}

{
    _id: 2,
    name: "pencil",
    price: 0.80
}

// Orders collection
{
    orderNumber: 3243,
    productsOrdered: [1,2]
}
```

Mongoose – Connecting node app to MongoDB

Connecting node app to mongo database via mongoose & adding a record in collection :-

Mongoose Schema :-

```
const schemaName = new mongoose.Schema({
    fieldName :  FieldDataType,
    …
})
```

Mongoose Model :-

```
const = mongoose.model(
   SingularCollectionName,
   schemaName
);
```

Mongoose Document/Record :-

```
const constantName = new ModelName ({
    fieldName :  fieldData,
    …
});
```

```
// Load mongoose package
const mongoose = require('mongoose');

// Specfify port of mongodb & database name
mongoose.connect("mongodb://localhost:27017/fruitsDB");

// Create schema for fruits collection
const fruitSchema = new mongoose.Schema({
  name: String,
  rating: Number
});

// Create mongoose model
// Specify collection name & schema to be followed by it
const Fruit = mongoose.model("Fruit", fruitSchema);

// Create New fruits record
const fruit = new Fruit ({name: "Apple", rating: 7});

// save fruit data in fruits collection & in fruitsdb
fruit.save();

// Close connection
mongoose.connection.close()
```

Mongoose – Performing CRUD

insertMany() :-

- For Adding multiple records in collection

- **Syntax :-**

```
ModelName.insertMany(RecordArray, function (err){
    //Deal with error or log success.
}
```

```
// Creating Fruits records
const kiwi = new Fruit({name: "kiwi", score: 10});
const orange = new Fruit({name: "Orange", score: 4});

// The insert func. Consists of array of records to be inserted
// error function to log if any errors occurred while inserting the records
Fruit.insertMany([kiwi, orange], function(err){
  if (err) {
    console.log(err)
  } else {
    console.log("Successfully saved all fruits to fruitsDB")
  }
});
```

Mongoose – Performing CRUD

Reading from database :-

- find() is used

- **Syntax :-**

```
ModelName.find( {conditions}, function (err, results){
    //Use the found results docs.
});
```

```
// Read all records present in Fruits collection
Fruit.find(function(err, fruits) {

  // Log if any errors occurred during finding records
  if (err) {
    console.log(err)
  } else {

    // close connection
    mongoose.connection.close()

    // log the fruit name from fruits array
    fruits.forEach(fruit => {
      console.log(fruit.name)
    });

  }
})
```

Mongoose – Performing CRUD

updateOne() :- • Update single record in collection	**// Error function to log if any errors occurred while updating the record** Fruit.updateOne({_id:"637d110dc9606c619709e5a0"}, {name: “peach"}, function(err){ if(err){ console.log(err); }else{ console.log("Successfully updated the document"); } })

Mongoose – Performing CRUD

deleteOne() :-

- Delete single record from collection

```
// Error function to log if any errors occurred while deleting the record
Fruit.deleteOne({name: 'peach'}, function(err) {
  if(err){
    console.log(err);
  }else{
    console.log("Successfully deleted the record");
  }
})
```

deleteMany() :-

- Delete multiple records from collection

```
// Error function to log if any errors occurred while deleting the record
// Delete all people entries having John name
People.deleteMany({name:"John"}, function(err) {
  if(err){
        console.log(err);
      }else{
        console.log("Successfully deleted the records");
      }
})
```

Mongoose – Performing CRUD

findByIdAndRemove() :-

- Remove record from collection by id

```
// Error function to log if any errors occurred while deleting the record
Fruit.findByIdAndRemove( _id, function(err) {
    if (err) {
      console.log(err)
    } else {
      console.log("Successfully deleted the record");
    }
})
```

Mongoose – Data validation

Data validation with Mongoose :-

- Data can be easily validated using mongoose data validation syntax
- https://mongoosejs.com/docs/validation.html

```
// Validate fruitSchema to have a required name
// & a rating between 1 to 10

const fruitSchema = new mongoose.Schema({

  name: {
    type: String,
    required: true
  },

  rating: {
    type: Number,
    min: 1,
    max: 10
  }

});
```

Mongoose – Implementing Relations

Implementing relations with mongoose :-	const kiwi = new Fruit({ name: "kiwi", score: 10, }); const person = new People({ name: "Amy", age: 12, favouriteFruit: kiwi });

Mongoose findOneAndUpdate()

```
<ModelName>.findOneAndUpdate(
  {conditions},
  {updates},
  function(err, results){}
);
```

```
<ModelName>.findOneAndUpdate(
  {conditions},
  {$pull: {field: {query}}},
  function(err, results){}
);
```

```
<ModelName>.findOneAndUpdate(
  {conditions},
  {$pull: {field: {_id: value}}},
  function(err, results){}
);
```