Databases

- Database is a collection of data.
- The data in the form of either a table or an object.

Types of Databases

1. Relational Databases or Structured Databases or SQL Databases
    - Engines: MySQL, PostgreSQL, SQLite, Oracle, SQL Server
2. Non-Relational Databases or NoSQL Databases
    - Engines: MongoDB, CouchDB, Cassandra, Redis, HBase, Elasticsearch

SQL Databases

- SQL stands for Structured Query Language.
- SQL is used to communicate with a database.
- SQL is used to perform tasks such as update data on a database, or retrieve data from a database.

SQL Commands

- DDL (Data Definition Language)
    - create, alter, drop, truncate of tables, databases, etc.
- DML (Data Manipulation Language)
    - insert, update, delete, select data from tables.
- DCL (Data Control Language)
    - grant, revoke, deny permissions to users.
- TCL (Transaction Control Language)
    - commit, rollback, savepoint, etc.

SQL Statements

- show all the databases

    ```sql
    show databases;
    ```

- create a database

    ```sql
      create database fsd56wedb;
    ```

- Show the database that is currently in use.

    ```sql
    select database();
    ```

- Use a database

```
  use fsd56wedb;
```

- Drop a database

```
  drop database fsd56wedb;
```

- create a table

```
create table products(
id int,
name varchar(100),
price float,
quantity int
);
```

- show the structure of the table

```
desc products;
```

- show all the tables in the database

```
show tables;
```

- drop a table

```
drop table products;
```

- view all the records in the table

```
select * from products;
```

- insert a record into the table

```
insert into products values
(1, 'iphone', 70000, 10),
(2, 'samsung', 50000, 20),
(3, 'oneplus', 40000, 30),
(4, 'mi', 20000, 40),
(5, 'realme', 15000, 50),
(6, 'oppo', 30000, 60),
(7, 'vivo', 25000, 70),
(8, 'nokia', 10000, 80),
(9, 'sony', 35000, 90),
(10, 'lg', 45000, 100),
(11, 'htc', 60000, 110),
(12, 'motorola', 80000, 120),
(13, 'lenovo', 7000, 130),
(14, 'asus', 9000, 140),
(15, 'google', 100000, 150);
```

- update a record in the table

```
  update products set price = 75000 where id = 1;
```

- alter a table

```
  alter table products add column description text;
```

- select specific columns from the table

```
  select id, name from products;
```

- select specific columns from the table where a condition is met

```
  select id, name from products where price > 50000;
```

- select specific columns from the table where a condition is met and order the results

```
  select id, name from products where price > 50000 order by price
desc;
```

- select specific columns from the table where a condition is met and order the results and limit the results

```
  select id, name from products where price > 50000 order by price
desc limit 3;
```

- select specific columns from the table where a condition is met and order the results and limit the results and offset the results

  ```
  select id, name from products where price > 50000 order by price desc limit 3 offset 2;
  ```

Normalization

Normalization is a process of organizing the data in the database efficiently, reducing redundancy and dependency by dividing the larger table into smaller tables and defining relationships between them.

Example:

books table

id	title	author	location	price
1	DSA	John	India	500
2	DBMS	Alice	USA	600
3	OS	Bob	UK	1300
4	CN	John	India	700
5	CO	Alice	USA	800
6	TOC	Bob	UK	1200
7	AI	John	India	1000
8	ML	Alice	USA	900
9	DL	Bob	UK	1100
10	DAA	John	India	300

The above table is not normalized because the author name and his location is repeated multiple times.

To normalize the above table, we can divide the table into two tables.

books table

id	title	price	author_id
1	DSA	500	1
2	DBMS	600	2
3	OS	1300	3
4	CN	700	1

id	title	price	author_id
5	CO	800	2
6	TOC	1200	3
7	AI	1000	1
8	ML	900	2
9	DL	1100	3
10	DAA	300	1

```
create table books(
id int,
title varchar(100),
price float,
author_id int
);

insert into books values
(1, 'DSA', 500, 1),
(2, 'DBMS', 600, 2),
(3, 'OS', 1300, 3),
(4, 'CN', 700, 1),
(5, 'CO', 800, 2),
(6, 'TOC', 1200, 3),
(7, 'AI', 1000, 1),
(8, 'ML', 900, 2),
(9, 'DL', 1100, 3),
(10, 'DAA', 300, 1);
```

authors table

id	name	location
1	John	India
2	Alice	USA
3	Bob	UK

```
create table authors(
id int,
name varchar(100),
location varchar(100)
);

insert into authors values
(1, 'John', 'India'),
(2, 'Alice', 'USA'),
(3, 'Bob', 'UK');
```

Now, the data is organized efficiently and redundancy is reduced.

How to get the data from the above two tables?

Joins

- Inner Join: Returns records that have matching values in both tables.
- Outer Join
  - Left Outer Join: Returns all records from the left table and the matched records from the right table.
  - Right Outer Join: Returns all records from the right table and the matched records from the left table.
  - Full Outer Join: Returns all records when there is a match in either left (table1) or right (table2) table records.

```
create table students(id int, name varchar(20));
```

```
create table enrollments(studentid int, course varchar(20));
```

```
insert into students values
(1, 'Alice'),
(2, 'Bob'),
(3, 'Charlie'),
(4, 'David'),
(5, 'Eve');
```

```
insert into enrollments values
(2, 'Maths'),
(3, 'Science'),
(4, 'English'),
(5, 'History'),
(6, 'Geography');
```

```
select * from students;
select * from enrollments;
```

Inner Join

```
select students.name, enrollments.course from students inner join
enrollments on students.id = enrollments.studentid;
```

Left Join

```
select students.name, enrollments.course from students left join
enrollments on students.id = enrollments.studentid;
```

Right Join

```
select students.name, enrollments.course from students right join
enrollments on students.id = enrollments.studentid;
```

Full Outer Join

```
select students.name, enrollments.course from students left outer join
enrollments on students.id = enrollments.studentid
union
select students.name, enrollments.course from students right outer join
enrollments on students.id = enrollments.studentid;
```

DB Model Design

Example: E-commerce website

User Stories:

1. As a user, I should be able to register on the website.
2. As a user, I should be able to login to the website.
3. As a user, I should be able to view all the products.
4. As a user, I should be able to view the details of a product.
5. As a user, I should be able to add a product to the cart.
6. As a user, I should be able to view the cart.
7. As a user, I should be able to remove a product from the cart.
8. As a user, I should be able to place an order.
9. As a user, I should be able to view all the orders.
10. As a user, I should be able to view the details of an order.

Approach:

1. Identify the entities
    - User
    - Product
    - Cart
    - Order
2. Identify the attributes
    - User: id, name, email, password
    - Product: id, name, price, quantity
    - Cart: id, userid, productid, quantity
    - Order: id, userid, productid, quantity, totalprice
3. Identify the relationships
    - User has many products
    - Product belongs to a user
    - User has one cart
    - Cart belongs to a user
    - Cart has many products
    - Product belongs to a cart
    - User has many orders
    - Order belongs to a user
    - User has many orders
    - Order belongs to a user
4. Identify the cardinality
    - User has many products (1 to many)
    - Product belongs to a user (many to 1)
    - User has one cart (1 to 1)
    - Cart belongs to a user (1 to 1)
    - Cart has many products (1 to many)
    - Product belongs to a cart (many to 1)
    - User has many orders (1 to many)
    - Order belongs to a user (many to 1)
    - User has many orders (1 to many)
    - Order belongs to a user (many to 1)
5. Identify the primary keys
    - User: id
    - Product: id
    - Cart: id
    - Order: id
6. Identify the foreign keys

- Product: userid
- Cart: userid, productid
- Order: userid, productid

7. Create the tables

```
create table users(
id int primary key auto_increment,
name varchar(100),
email varchar(100),
password varchar(100)
);

create table products(
id int primary key auto_increment,
name varchar(100),
price float,
quantity int,
userid int,
foreign key(userid) references users(id)
);

create table carts(
id int primary key auto_increment,
userid int,
productid int,
quantity int,
foreign key(userid) references users(id),
foreign key(productid) references products(id)
);

create table orders(
id int primary key auto_increment,
userid int,
productid int,
quantity int,
totalprice float,
foreign key(userid) references users(id),
foreign key(productid) references products(id)
);
```

8. Insert the data

```
insert into users values
(1, 'Alice', 'alice@xyz.com', 'alice123'),
(2, 'Bob', 'bob@xyz.com', 'bob123'),
(3, 'Charlie', 'charlie@xyz.com', 'charlie123'),
(4, 'David', 'david@xyz.com', 'david123'),
(5, 'Eve', 'eve@xyz.com', 'eve123');
```

```
insert into products values
(1, 'iphone', 70000, 10, 1),
(2, 'samsung', 50000, 20, 2),
(3, 'oneplus', 40000, 30, 3),
(4, 'mi', 20000, 40, 4),
(5, 'realme', 15000, 50, 5);
```

```
insert into carts values
(1, 1, 1, 1),
(2, 2, 2, 2),
(3, 3, 3, 3),
(4, 4, 4, 4),
(5, 5, 5, 5);
```

```
insert into orders values
(1, 1, 1, 1, 70000),
(2, 2, 2, 2, 100000),
(3, 3, 3, 3, 120000),
(4, 4, 4, 4, 80000),
(5, 5, 5, 5, 75000);
```

Questions:

1. Which user has placed maximum orders?

MongoDB- Day -1 : Database - MongoDB

Contents:

✅ Intro to MongoDB
✅ Why Mongodb?
✅ what is a document?
✅ what is a collection?
✅ MongoDB vs MySQL
✅ installation of MongoDB
✅ creation of database, ✅ creation of collections, ✅ creation of documents
✅ find - query & projection

Task

https://docs.google.com/document/d/1yfQTicBMNV7thyzewyYSJWhTNw0pC4AkVhMGVCIP0LU/edit

MongoDB

- MongoDB is a NoSQL database.

- MongoDB is a document-oriented database.
- MongoDB is an unstructured database.

What is a Document?

- A document is a record in MongoDB.
- A document is a JSON object.
- A document is a key-value pair.
- A document is a record in a table in SQL databases.

What is a Collection?

- A collection is a group of documents.
- A collection is a table in SQL databases.

Why MongoDB?

- MongoDB is easier to scale compared to SQL databases.
- MongoDB is faster than SQL databases.
- MongoDB is unstructured which means we can store any type of data in MongoDB.

MongoDB vs MySQL

MySQL

- MySQL is a relational database.
- MySQL is a structured database.
- MySQL is a table-based database.

MongoDB

- MongoDB is a NoSQL database.
- MongoDB is an unstructured database.
- MongoDB is a document-oriented database.

MongoDB Queries

List all the databases

```
show dbs;
```

or

```
show databases;
```

To see the current database

```
db;
```

To switch to a database or to create a database if it does not exist

```
use fsd56wedb;
```

To drop a database

```
db.dropDatabase();
```

Note: After dropping a database, switch to another database.

To create a collection

```
db.createCollection('products');
```

To insert a document into a collection

```
db.products.insertOne({
id: 1,
name: 'iphone',
price: 70000,
quantity: 10
})
```

or

```
db.products.insert({
id: 1,
name: 'iphone',
price: 70000,
quantity: 10
})
```

To insert multiple documents into a collection

```
db.products.insertMany([
  {
    id: 2,
    name: 'samsung',
    price: 50000,
    quantity: 20
  },
  {
    id: 3,
    name: 'oneplus',
    price: 40000,
    quantity: 30
  },
  {
    id: 4,
    name: 'mi',
    price: 20000,
    quantity: 40
  },
  {
    id: 5,
    name: 'realme',
    price: 15000,
    quantity: 50
  }
])
```

To delete a document from a collection

```
db.products.deleteOne({id: 1})
```

Note: It will delete the first document that matches the query.

To delete multiple documents from a collection

```
db.products.deleteMany({price: 50000})
```

To view all the documents in a collection

```
db.products.find();
```

To view all the documents in a collection based on a condition

```
db.products.find({price: 70000});
```

To view all the documents in a collection based on the projection

```
db.products.find({}, {id: 1, name: 1});
```

Note: this will include _id, id, name in the resulting table and excludes price, quantity.

```
db.products.find({}, {id: 1, name: 1, _id: 0});
```

Note: this will include id, name in the resulting table and excludes price, quantity, _id.

```
db.products.find({}, {id: 0, name: 0});
```

Note: this will exclude id, name but includes price, quantity, _id.

MongoDB- Day -2 : Database - MongoDB

Contents:

- [x] use of operators in find()
- [x] basic cursor methods - map, toArray, pretty, forEach, limit, count, sort
- [ ] Aggregation

Use of operators in find()

- $lt: less than
- $lte: less than or equal to
- $gt: greater than
- $gte: greater than or equal to
- $eq: equal to
- $ne: not equal to
- $in: in
- $nin: not in
- $or: or
- $and: and
- $not: not

```
db.products.find({price: {$lt: 50000}});
db.products.find({price: {$lte: 50000}});
```

```
db.products.find({price: {$gt: 50000}});
```

```
db.products.find({price: {$gte: 50000}});
```

```
db.products.find({price: {$eq: 50000}});
```

```
db.products.find({price: {$ne: 50000}});
```

```
db.products.find({price: {$in: [50000, 40000]}});
```

```
db.products.find({price: {$nin: [50000, 40000]}});
```

```
db.products.find({$or: [{price: 50000}, {price: 40000}]});
```

```
db.products.find({$and: [{price: 50000}, {quantity: 20}]});
```

```
db.products.find({$not: {price: 50000}});
```

Basic Cursor Methods

Cursor:

- A pointer to the result set of a query.
- MongoDB returns a cursor when the find() method is called.
- It is used to iterate over the result set.
- Once after the iteration, the cursor is exhausted or closed.

```
let products = db.products.find().toArray();
```

```
products.forEach(doc => print(doc));
```

```
products.forEach(doc => print(doc.price));
```

```
products.forEach((doc) => { if (doc.price >= 50000) print(doc); });
```

```
 products.map(doc => [doc.name, doc.price, doc.quantity])
```

```
db.products.find().sort()
```

Sort by price in ascending order

```
db.products.find().sort({
... price: 1
... });
```

Sort by price in descending order

```
db.products.find().sort({
... price: -1
... });
```

Sort by price and quantity in descending order

```
db.products.find().sort({
... price: -1,
... quantity: -1
... });
```

Limit the results

```
db.products.find().limit(3);
```

Skip the results and limit the results

```
db.products.find().skip(2).limit(3);
```