1. Create a table and insert minimum 5 records For Facebook/ Adhar and draw ER diagram. Demonstrate what is Hadoop and HBase

```
-- Create table for Facebook and Aadhaar records

CREATE TABLE Users (

  id INT PRIMARY KEY,

  name VARCHAR(100),

  email VARCHAR(100),

  dob DATE,

  aadhaar_number VARCHAR(12)

);


-- Inserting 5 records

INSERT INTO Users (id, name, email, dob, aadhaar_number) VALUES

(1, 'Raj Kumar', 'rajkumar@example.com', '1990-06-15', '1234 5678 9012'),

(2, 'Anjali Mehta', 'anjali@example.com', '1992-08-22', '2345 6789 0123');
```

2. Create a table and insert minimum 5 records For Twitter /Flipkart and draw ER diagram. Discuss about Hive and

Flipkart Table

This table will store basic information about products listed on Flipkart.

```
sql
Copy code
-- Create table for Flipkart products
CREATE TABLE FlipkartProducts (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10, 2),
    stock_quantity INT
);

-- Inserting 5 records
INSERT INTO FlipkartProducts (product_id, product_name, category, price,
stock_quantity) VALUES
(1, 'Apple iPhone 14', 'Electronics', 74999.99, 150),
(2, 'Samsung Galaxy S21', 'Electronics', 59999.99, 200),
(3, 'BoAt Rockerz 450', 'Accessories', 1499.99, 500),
(4, 'Dell Inspiron 15', 'Laptops', 54999.99, 100),
(5, 'Nike Running Shoes', 'Footwear', 3999.99, 250);
```

ER Diagram for Twitter and Flipkart

ER Diagram for Twitter:

- **Entity**: TwitterUser

- Attributes:
  - `user_id` (Primary Key)
  - `username`
  - `email`
  - `date_of_birth`
  - `followers_count`

ER Diagram for Flipkart:

- **Entity**: FlipkartProduct
  - Attributes:
    - `product_id` (Primary Key)
    - `product_name`
    - `category`
    - `price`
    - `stock_quantity`

The relationships here could be:

- **For Twitter**: One user can have many followers, but we are not storing detailed follower information in this table.
- **For Flipkart**: Each product can belong to a single category (One-to-Many relation between categories and products).
-

3. Create a table For Facebook/ Adhar and perform following operations 1.Insert 2.Select 3.Delete

Create a Table for Facebook/Aadhaar

```
-- Inserting records into the Users table
INSERT INTO Users (id, name, email, dob, aadhaar_number) VALUES
(1, 'Raj Kumar', 'rajkumar@example.com', '1990-06-15', '1234 5678 9012'),
(2, 'Anjali Mehta', 'anjali@example.com', '1992-08-22', '2345 6789 0123'),
(3, 'Sandeep Singh', 'sandeep@example.com', '1989-03-10', '3456 7890 1234'),
(4, 'Priya Sharma', 'priya@example.com', '1995-12-01', '4567 8901 2345'),
(5, 'Rahul Kumar', 'rahul@example.com', '1993-07-19', '5678 9012 3456');
```

select

```
-- Selecting all records from the Users table
SELECT * FROM Users;
-- Selecting name and email for user with id 1
SELECT name, email FROM Users WHERE id = 1;
```

Delete

```
-- Deleting a record where the user_id is 3
DELETE FROM Users WHERE id = 3;
```

4. Create a table For Twitter /Flipkart and perform following Operations 1.Insert 2.Update 3.Drop

```
-- Create table for Twitter users
CREATE TABLE TwitterUsers (
   user_id INT PRIMARY KEY,          -- Unique ID for each user
   username VARCHAR(50),             -- Twitter username
   email VARCHAR(100),               -- User's email address
   followers_count INT               -- Number of followers
);

-- Inserting records into FlipkartProducts table
INSERT INTO FlipkartProducts (product_id, product_name, category, price, stock_quantity)
VALUES
(1, 'Apple iPhone 14', 'Electronics', 74999.99, 150),
(2, 'Samsung Galaxy S21', 'Electronics', 59999.99, 200),
(3, 'BoAt Rockerz 450', 'Accessories', 1499.99, 500),
(4, 'Dell Inspiron 15', 'Laptops', 54999.99, 100),
(5, 'Nike Running Shoes', 'Footwear', 3999.99, 250);

-- Update the price for product_id 3 (BoAt Rockerz 450)
UPDATE FlipkartProducts
SET price = 1699.99
WHERE product_id = 3;

drop
-- Drop the FlipkartProducts table
DROP TABLE FlipkartProducts;
```

``````````````

**5. Create a table For Facebook/ Adhar and perform following operations**
**1.Update 2.Aggregate function (Max, Min,Count) 3.Join operations (Inner Join ,Left Join,Right Join)**

```
-- Create table for Facebook and Aadhaar users
CREATE TABLE FacebookAadhaarUsers (
   user_id INT PRIMARY KEY,          -- Unique ID for each user
   name VARCHAR(100),                -- User's full name
   email VARCHAR(100),               -- User's email address
   dob DATE,                         -- User's date of birth
   aadhaar_number VARCHAR(12)        -- Unique Aadhaar number
);
-- Inserting records into the FacebookAadhaarUsers table
INSERT INTO FacebookAadhaarUsers (user_id, name, email, dob, aadhaar_number) VALUES
(1, 'Raj Kumar', 'rajkumar@example.com', '1990-06-15', '1234 5678 9012'),
```
```

(2, 'Anjali Mehta', 'anjali@example.com', '1992-08-22', '2345 6789 0123'),
(3, 'Sandeep Singh', 'sandeep@example.com', '1989-03-10', '3456 7890 1234'),
(4, 'Priya Sharma', 'priya@example.com', '1995-12-01', '4567 8901 2345'),
(5, 'Rahul Kumar', 'rahul@example.com', '1993-07-19', '5678 9012 3456');

Update Operation
-- Update the email address for user_id = 3
UPDATE FacebookAadhaarUsers
SET email = 'sandeep_new@example.com'

**WHERE user_id = 3;**

## Aggregate Functions

Now, let's use aggregate functions to perform:

- **Max**: Find the latest date of birth (maximum value).
- **Min**: Find the earliest date of birth (minimum value).
- **Count**: Find the number of users in the table.

**Max: Latest Date of Birth**
-- Find the latest date of birth

SELECT MAX(dob) AS latest_dob FROM FacebookAadhaarUsers;

**MIN**
-- Find the earliest date of birth

SELECT MIN(dob) AS earliest_dob FROM FacebookAadhaarUsers;

**count**
-- Find the total number of users

## . Join Operations

Let's assume we have another table `UserAddresses` that stores the addresses of the users, with fields like:

- `user_id`: Unique identifier for each user.
- `address`: User's address.

**UserAddresses Table Creation**

-- Create a table for User Addresses

CREATE TABLE UserAddresses (

user_id INT PRIMARY KEY, -- Unique ID for each user

address VARCHAR(255), -- User's address

FOREIGN KEY (user_id) REFERENCES FacebookAadhaarUsers(user_id)

**);**

**Inner Join**

An **Inner Join** returns only the rows where there is a match in both tables. Let's join `FacebookAadhaarUsers` with `UserAddresses` on `user_id`.

```
sql
Copy code
-- Perform an Inner Join between FacebookAadhaarUsers and UserAddresses
SELECT u.user_id, u.name, u.email, a.address
FROM FacebookAadhaarUsers u
INNER JOIN UserAddresses a ON u.user_id = a.user_id;
```

**Left Join**

A **Left Join** returns all rows from the left table (`FacebookAadhaarUsers`), and the matching rows from the right table (`UserAddresses`). If there is no match, the result is `NULL` on the right side.

```
sql
Copy code
-- Perform a Left Join between FacebookAadhaarUsers and UserAddresses
SELECT u.user_id, u.name, u.email, a.address
FROM FacebookAadhaarUsers u
LEFT JOIN UserAddresses a ON u.user_id = a.user_id;
```

**Right Join**

A **Right Join** returns all rows from the right table (`UserAddresses`), and the matching rows from the left table (`FacebookAadhaarUsers`). If there is no match, the result is `NULL` on the left side.

```
sql
Copy code
-- Perform a Right Join between FacebookAadhaarUsers and UserAddresses
SELECT u.user_id, u.name, u.email, a.address
FROM FacebookAadhaarUsers u
RIGHT JOIN UserAddresses a ON u.user_id = a.user_id;
```

## 6. Create a table For Twitter /Flipkart and perform following operations

## 1.Alter 2.Aggregate function (Sum,Avg,Min) 3.Join

## operations(Full Join,Inner Join)

**-- Create a table for Flipkart products**

CREATE TABLE FlipkartProducts (

product_id INT PRIMARY KEY, -- Unique ID for each product

```
    product_name VARCHAR(100),        -- Product name
    category VARCHAR(50),             -- Product category
    price DECIMAL(10, 2),             -- Product price
    stock_quantity INT                -- Available stock quantity
);
-- Inserting records into FlipkartProducts table
INSERT INTO FlipkartProducts (product_id, product_name, category, price, stock_quantity) VALUES
(1, 'iPhone 14', 'Electronics', 74999.99, 150),
(2, 'Samsung Galaxy S21', 'Electronics', 59999.99, 200),
(3, 'BoAt Rockerz 450', 'Accessories', 1499.99, 500),
(4, 'Dell Inspiron 15', 'Laptops', 54999.99, 100),
(5, 'Nike Running Shoes', 'Footwear', 3999.99, 250);
```

. **Aggregate Functions**

```
-- Average price of products in Flipkart
SELECT AVG(price) AS average_price FROM FlipkartProducts;
```

### Min: Minimum Stock Quantity in Flipkart

```
sql
Copy code
-- Minimum stock quantity of products in Flipkart
SELECT MIN(stock_quantity) AS min_stock_quantity FROM FlipkartProducts;
```

## 5. Join Operations

Let's perform two types of **Join Operations** between the **TwitterUsers** and **FlipkartProducts** tables:

- **Full Join**: Retrieves all rows from both tables, with matching rows from both sides. If no match is found, it returns `NULL`.
- **Inner Join**: Retrieves only the rows where there is a match in both tables.

### Full Join

A **Full Join** retrieves all records from both tables, matching records where possible. If there is no match, it returns `NULL` values on the side that does not have a match.

# 7. Write a function to update particular value in table and write a Stored Procedure to insert value in table (Assume any Table)

```
-- Perform a Full Join between TwitterUsers and FlipkartProducts
```

```
SELECT u.user_id, u.username, p.product_name, p.price
FROM TwitterUsers u
FULL OUTER JOIN FlipkartProducts p
ON u.user_id = p.product_id;
```

**Note**: Some SQL systems (e.g., MySQL) do not directly support `FULL OUTER JOIN`. If needed, we can simulate it using `LEFT JOIN` and `RIGHT JOIN` together.

**Inner Join**

An **Inner Join** returns only the rows where there is a match in both tables.

```
-- Perform an Inner Join between TwitterUsers and FlipkartProducts
SELECT u.user_id, u.username, p.product_name, p.price
FROM TwitterUsers u
INNER JOIN FlipkartProducts p
ON u.user_id = p.product_id;
```

## Summary of Operations:

1. **Alter Table**: Modify the table structure (e.g., add or modify columns).
2. **Aggregate Functions**:
   - **Sum**: Total of a specified column.
   - **Avg**: Average value of a specified column.
   - **Min**: Minimum value of a specified column.
3. **Join Operations**:
   - **Full Join**: Retrieves all rows from both tables, including unmatched rows.
   - **Inner Join**: Retrieves only the rows where there is a match in both tables.

**7. Write a function to update particular value in table and write a Stored Procedure to insert value in table (Assume any Table)**

```
-- Create Products table
CREATE TABLE Products (
   product_id INT PRIMARY KEY,       -- Unique ID for each product
   product_name VARCHAR(100),        -- Product name
   category VARCHAR(50),             -- Product category
   price DECIMAL(10, 2),             -- Product price
   stock_quantity INT                -- Quantity available in stock

);
```

Let's create a function and a stored procedure using SQL. We will assume a simple **Products** table for this demonstration.

## 1. Create a Table (Products)

For the sake of the demonstration, we'll assume a table `Products` with the following columns:

- `product_id`: Unique identifier for the product.
- `product_name`: Name of the product.
- `category`: Category of the product.
- `price`: Price of the product.
- `stock_quantity`: Quantity available in stock.

### Creating the Products Table

```
sql
Copy code
-- Create Products table
CREATE TABLE Products (
    product_id INT PRIMARY KEY,        -- Unique ID for each product
    product_name VARCHAR(100),         -- Product name
    category VARCHAR(50),              -- Product category
    price DECIMAL(10, 2),              -- Product price
    stock_quantity INT                -- Quantity available in stock
);
```

## 2. Function to Update a Particular Value in the Table

Let's create a function that allows us to update a particular value in the table, for example, updating the `price` of a specific product by `product_id`.

### SQL Function to Update Price

The function will take two parameters: `product_id` (the ID of the product) and `new_price` (the new price to be updated).

```
sql
Copy code
-- Create a function to update the price of a product based on product_id
CREATE FUNCTION update_product_price (prod_id INT, new_price DECIMAL(10,
2))
RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
    -- Update the product price in the Products table
    UPDATE Products
    SET price = new_price
    WHERE product_id = prod_id;

    -- Return a success message
    RETURN 'Product price updated successfully';
END;
```

### Example of Using the Function

```
sql
Copy code
-- Call the function to update the price of a product with product_id 1 to
1999.99
SELECT update_product_price(1, 1999.99);
```

## 3. Stored Procedure to Insert a Value in the Table

Now, let's write a **stored procedure** to insert a new product into the `Products` table.

**SQL Stored Procedure to Insert a Product**

The stored procedure will take parameters for `product_id`, `product_name`, `category`, `price`, and `stock_quantity`, and insert them into the table.

```
sql
Copy code
-- Create a stored procedure to insert a new product
DELIMITER $$

CREATE PROCEDURE insert_product (
    IN prod_id INT,
    IN prod_name VARCHAR(100),
    IN prod_category VARCHAR(50),
    IN prod_price DECIMAL(10, 2),
    IN prod_stock INT
)
BEGIN
    -- Insert a new product into the Products table
    INSERT INTO Products (product_id, product_name, category, price,
stock_quantity)
    VALUES (prod_id, prod_name, prod_category, prod_price, prod_stock);

    -- Optional: Return a message (you can use SELECT or SIGNAL to return
messages)
    SELECT 'Product inserted successfully' AS message;
END $$

DELIMITER ;
```

**Example of Using the Stored Procedure**

To insert a new product into the `Products` table, you would call the stored procedure like this:

```
sql
Copy code
-- Call the stored procedure to insert a new product
CALL insert_product(6, 'OnePlus 11', 'Electronics', 49999.99, 100);
```

## Summary:

- **Function to Update Value**: The `update_product_price` function takes the product ID and new price as parameters and updates the price in the `Products` table.
- **Stored Procedure to Insert Value**: The `insert_product` stored procedure takes multiple parameters (product details) and inserts them into the `Products` table.

## 8. Write a trigger to insert , update record from Library

system 8table

## 1. Create a Library System Table

First, we need a table that represents the Library system. We'll create a table called `Books` with the following columns:

- `book_id`: Unique ID for each book.
- `title`: Title of the book.
- `author`: Author of the book.
- `category`: Book category (e.g., Fiction, Non-Fiction).
- `price`: Price of the book.
- `stock_quantity`: Quantity available in stock.

### SQL: Creating the `Books` Table

```
sql
Copy code
-- Create the Books table for the Library System
CREATE TABLE Books (
    book_id INT PRIMARY KEY,           -- Unique ID for each book
    title VARCHAR(100),                -- Title of the book
    author VARCHAR(100),               -- Author of the book
    category VARCHAR(50),              -- Category of the book (Fiction,
Non-Fiction, etc.)
    price DECIMAL(10, 2),              -- Price of the book
    stock_quantity INT                -- Available stock of the book
);
```

## 2. Trigger for Insert and Update Operations

We will create a trigger that:

- **After Insert**: Logs the insertion of a new book.
- **After Update**: Logs the update of any book's details.

### SQL: Creating the Trigger

The trigger will use a **Log** table to store the operations performed on the `Books` table.

```
sql
Copy code
-- Create a Log table to store actions (insert and update) on the Books
table
CREATE TABLE BooksLog (
    log_id INT AUTO_INCREMENT PRIMARY KEY,  -- Unique ID for each log
    action VARCHAR(50),                     -- Action performed (Insert or
Update)
    book_id INT,                            -- ID of the book
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- Time of the action
    details VARCHAR(255)                    -- Details of the action
(Old/Updated values)
);

-- Create the trigger that is activated after an Insert operation on the
Books table
DELIMITER $$

CREATE TRIGGER after_insert_books
```

```
AFTER INSERT ON Books
FOR EACH ROW
BEGIN
    -- Insert a record into BooksLog after a new book is inserted
    INSERT INTO BooksLog (action, book_id, details)
    VALUES ('INSERT', NEW.book_id, CONCAT('New book added: ', NEW.title, '
by ', NEW.author));
END $$

-- Create the trigger that is activated after an Update operation on the
Books table
CREATE TRIGGER after_update_books
AFTER UPDATE ON Books
FOR EACH ROW
BEGIN
    -- Insert a record into BooksLog after a book's details are updated
    INSERT INTO BooksLog (action, book_id, details)
    VALUES ('UPDATE', OLD.book_id, CONCAT('Book updated: ', OLD.title, '
changed to ', NEW.title));
END $$

DELIMITER ;
```

## 3. Explanation of the Trigger

1. **Trigger for Insert**:
    - **Name**: `after_insert_books`
    - **When**: This trigger will be activated **after** a new record is inserted into the `Books` table.
    - **Action**: It inserts a log record into the `BooksLog` table with the action type (`INSERT`), the `book_id`, and details of the new book added (title and author).
2. **Trigger for Update**:
    - **Name**: `after_update_books`
    - **When**: This trigger will be activated **after** an existing record in the `Books` table is updated.
    - **Action**: It inserts a log record into the `BooksLog` table with the action type (`UPDATE`), the `book_id`, and details of the change (old and new titles).

## 4. Example: Inserting and Updating Books

### Inserting a New Book

```
sql
Copy code
-- Insert a new book into the Books table
INSERT INTO Books (book_id, title, author, category, price, stock_quantity)
VALUES (1, 'The Catcher in the Rye', 'J.D. Salinger', 'Fiction', 499.99,
10);
```

### Updating an Existing Book

```
sql
Copy code
-- Update a book's details (e.g., change the title of book_id 1)
UPDATE Books
```

```
SET title = 'The Catcher in the Rye (Updated Edition)'
WHERE book_id = 1;
```

## 5. Checking the Log Table

After performing insert and update operations, you can view the log of actions by querying the `BooksLog` table:

```
sql
Copy code
-- View the logs of actions on the Books table
SELECT * FROM BooksLog;
```

## Summary:

- **Triggers** are used to automatically perform actions in the database when certain events (like INSERT or UPDATE) occur.
- In this example:
  - An **INSERT** trigger logs a new book being added to the system.
  - An **UPDATE** trigger logs any changes made to a book's details.
- The **BooksLog** table stores logs about the operations performed, providing an audit trail.

## 9. Write a cursor to insert , delete record from Employee table

```
-- Create the Employee table

CREATE TABLE Employee (

   emp_id INT PRIMARY KEY,

   emp_name VARCHAR(100),

   emp_salary DECIMAL(10, 2),

   emp_department VARCHAR(50)

);



-- Create a stored procedure to insert and delete records using a cursor

DELIMITER $$



CREATE PROCEDURE insert_delete_employee()

BEGIN
```

```
DECLARE done INT DEFAULT 0;
DECLARE v_emp_id INT;
DECLARE v_emp_name VARCHAR(100);
DECLARE v_emp_salary DECIMAL(10, 2);
DECLARE v_emp_department VARCHAR(50);

-- Declare a cursor for inserting employee records
DECLARE insert_cursor CURSOR FOR
   SELECT 101, 'John Doe', 55000.00, 'IT' UNION ALL
   SELECT 102, 'Jane Smith', 60000.00, 'HR' UNION ALL
   SELECT 103, 'Tom Brown', 70000.00, 'Finance';

-- Declare a cursor for deleting employee records based on emp_id
DECLARE delete_cursor CURSOR FOR
   SELECT emp_id FROM Employee WHERE emp_id = 102;

-- Declare a continue handler for the insert_cursor to stop at the end
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

-- Open the insert_cursor to insert new employee records
OPEN insert_cursor;

read_loop: LOOP
   FETCH insert_cursor INTO v_emp_id, v_emp_name, v_emp_salary, v_emp_department;
```

```
    IF done THEN
        LEAVE read_loop;
    END IF;

    -- Insert the fetched data into the Employee table
    INSERT INTO Employee (emp_id, emp_name, emp_salary, emp_department)
    VALUES (v_emp_id, v_emp_name, v_emp_salary, v_emp_department);
END LOOP;

-- Close the insert_cursor
CLOSE insert_cursor;

-- Open the delete_cursor to delete the employee with emp_id = 102
OPEN delete_cursor;

delete_loop: LOOP
    FETCH delete_cursor INTO v_emp_id;

    IF done THEN
        LEAVE delete_loop;
    END IF;

    -- Delete the employee record with the fetched emp_id
```

```
        DELETE FROM Employee WHERE emp_id = v_emp_id;

    END LOOP;


    -- Close the delete_cursor

    CLOSE delete_cursor;

END $$


DELIMITER ;


-- Call the stored procedure to perform the insert and delete operations

CALL insert_delete_employee();
```

# 10.erform Create, Update, Delete operations in MongoDB

## 1. Create Operation (Insert)

To insert a document into a collection, you can use `insertOne()` or `insertMany()` in **mongosh**.

```
js
Copy code
// Switch to the 'company' database
use company;

// Insert a single document into the 'employees' collection
db.employees.insertOne({
    emp_id: 101,
    emp_name: "John Doe",
    emp_salary: 50000,
    emp_department: "IT"
});

// Insert multiple documents
db.employees.insertMany([
    { emp_id: 102, emp_name: "Jane Smith", emp_salary: 60000,
emp_department: "HR" },
    { emp_id: 103, emp_name: "Tom Brown", emp_salary: 70000,
emp_department: "Finance" }
]);
```

## 2. Update Operation

To update documents in **mongosh**, you can use `updateOne()`, `updateMany()`, or `replaceOne()`.

```
js
Copy code
// Update a single document with emp_id 101
db.employees.updateOne(
    { emp_id: 101 },  // Filter condition
    { $set: { emp_salary: 55000 } }  // Update operation
);

// Update multiple documents (e.g., salary increase for employees in IT
department)
db.employees.updateMany(
    { emp_department: "IT" },  // Filter condition
    { $set: { emp_salary: 60000 } }  // Update operation
);
```

## 3. Delete Operation

To delete a document, you can use `deleteOne()` or `deleteMany()`.

```
js
Copy code
// Delete a single document with emp_id 101
db.employees.deleteOne({ emp_id: 101 });

// Delete all employees in the 'HR' department
db.employees.deleteMany({ emp_department: "HR" });
```

## Summary for mongosh:

- **Create**: `insertOne()` or `insertMany()`
- **Update**: `updateOne()`, `updateMany()`
- **Delete**: `deleteOne()`, `deleteMany()`

## 11. Perform Create, Read, Delete operations in MongoDB

use myDatabase; // Switch to 'myDatabase'

db.createCollection("users"); // Explicitly create 'users' collection (optional)

db.users.insertOne({

name: "John Doe",

email: "john@example.com",

age: 30,

status: "active"

```
db.users.find();

db.users.find({ age: { $gt: 30 } });

db.users.findOne({ email: "john@example.com" });

db.users.deleteOne({ name: "John Doe" })

db.users.deleteMany({ status: "inactive" });

db.users.find();
```

**12.**

1. Create a Table as **employee** and the details are

| S.No | Name | Designation | Branch |
|---|---|---|---|
| 1 | Ram | Manager | Chennai |
| 2 | Santhosh | Supervisor | Madurai |
| 3 | Hari | Assistant | Trichy |

Perform the following:

- Alter the table by adding a column **Salary**
- Alter the table by modifying the column **Name**
- Describe the table **employee**
- Copy the table **employee** as **emp**
- Truncate the table
- Delete the Second row from the table
- Drop the table

Here is the **SQL code** to perform the tasks mentioned in the image:

---

## 1. Create the `employee` table

```
sql
Copy code
CREATE TABLE employee (
    SNo INT PRIMARY KEY,
    Name VARCHAR(50),
    Designation VARCHAR(50),
    Branch VARCHAR(50)
);

INSERT INTO employee (SNo, Name, Designation, Branch)
VALUES
    (1, 'Ram', 'Manager', 'Chennai'),
    (2, 'Santhosh', 'Supervisor', 'Madurai'),
    (3, 'Hari', 'Assistant', 'Trichy');
```

---

## 2. Alter the table by adding a column `Salary`

```
sql
Copy code
ALTER TABLE employee ADD Salary DECIMAL(10, 2);
```

---

## 3. Alter the table by modifying the column `Name`

```
sql
Copy code
ALTER TABLE employee MODIFY Name VARCHAR(100);
```

---

## 4. Describe the table `employee`

To describe the structure of the table:

```
sql
Copy code
DESC employee;
```

---

## 5. Copy the table `employee` as `emp`

```
sql
Copy code
CREATE TABLE emp AS SELECT * FROM employee;
```

---

## 6. Truncate the table

To remove all data from the table:

```
sql
Copy code
TRUNCATE TABLE employee;
```

---

## 7. Delete the second row from the table

Before truncating, you can delete a specific row:

```
sql
Copy code
DELETE FROM employee WHERE SNo = 2;
```

---

## 8. Drop the table

To permanently remove the table:

```
sql
Copy code
DROP TABLE employee;
```

---

### Steps to Execute:

- Run each query in order in your SQL environment (e.g., MySQL Workbench, phpMyAdmin, or a command-line interface connected to the database).
- Ensure that the database you are working with has sufficient privileges for these operations.

# 13.

1. Create a Table as **bank** and the details are

| S.No | Cust_Name | Acc_no | Balance | Cus_Branch |
|---|---|---|---|---|
| 1 | Ramesh | 12378 | 100000 | Adyar |
| 2 | Sam | 12367 | 152500 | Mylapore |
| 3 | Harish | 12345 | 250000 | Anna Salai |

Perform the following:

- Simple Select
- Select with **where** clause
- Select with **comparison operator >**
- Select with **between** in the field **Balance**
- Update the **Cus_Branch** in the second row as **Poonamallee**

## 1. Create the `bank` table

```
sql
Copy code
CREATE TABLE bank (
    SNo INT PRIMARY KEY,
    Cust_Name VARCHAR(50),
    Acc_no INT,
    Balance DECIMAL(15, 2),
    Cus_Branch VARCHAR(50)
);

INSERT INTO bank (SNo, Cust_Name, Acc_no, Balance, Cus_Branch)
VALUES
    (1, 'Ramesh', 12378, 100000, 'Adyar'),
    (2, 'Sam', 12367, 152500, 'Mylapore'),
    (3, 'Harish', 12345, 250000, 'Anna Salai');
```

---

## 2. Simple `SELECT`

To select all records:

```
sql
Copy code
SELECT * FROM bank;
```

---

### 3. Select with `WHERE` clause

To find a customer with the branch "Adyar":

```
sql
Copy code
SELECT * FROM bank WHERE Cus_Branch = 'Adyar';
```

---

### 4. Select with `comparison operator >`

To find customers with a balance greater than 150000:

```
sql
Copy code
SELECT * FROM bank WHERE Balance > 150000;
```

---

### 5. Select with `BETWEEN`

To find customers with a balance between 120000 and 200000:

```
sql
Copy code
SELECT * FROM bank WHERE Balance BETWEEN 120000 AND 200000;
```

---

### 6. Update the `Cus_Branch` in the second row

To update the branch of the customer "Sam" to "Poonamallee":

```
sql
Copy code
UPDATE bank
SET Cus_Branch = 'Poonamallee'
WHERE SNo = 2;
```

---

### Steps to Execute:

1. Create the table and insert data using the `CREATE TABLE` and `INSERT` statements.
2. Run the queries in your SQL environment to perform the required operations.
3. Verify the results after each step using the `SELECT` statements.

## 14. Implement MapReduce in MongoDB with suitable dataset

**mplementing MapReduce in MongoDB**

MapReduce in MongoDB is used for aggregation operations, processing large datasets, and generating summarized results. Below is an example with a **sales dataset** to calculate the total sales for each product using MapReduce.

---

## Dataset

Let's create a `sales` collection:

```
javascript
Copy code
db.sales.insertMany([
    { product: "Laptop", amount: 1000, quantity: 2 },
    { product: "Mobile", amount: 500, quantity: 5 },
    { product: "Laptop", amount: 1200, quantity: 1 },
    { product: "Tablet", amount: 800, quantity: 3 },
    { product: "Mobile", amount: 600, quantity: 4 },
    { product: "Tablet", amount: 900, quantity: 2 }
]);
```

---

## MapReduce Example

1. **Map Function**: The `map` function emits key-value pairs, where the key is the product, and the value is the total sale (`amount * quantity`).
2. **Reduce Function**: The `reduce` function calculates the total sales for each product by summing up the values.

### Code:

```
javascript
Copy code
// Map Function
var mapFunction = function () {
    emit(this.product, this.amount * this.quantity);
};

// Reduce Function
var reduceFunction = function (key, values) {
    return Array.sum(values);
};

// Execute MapReduce
db.sales.mapReduce(
    mapFunction,
    reduceFunction,
    {
        out: "sales_summary" // Output collection
    }
);

// View Results
db.sales_summary.find();
```

---

## Expected Output

After executing the `mapReduce`, the `sales_summary` collection will have the following results:

```
javascript
Copy code
{ "_id": "Laptop", "value": 3200 }
{ "_id": "Mobile", "value": 4400 }
{ "_id": "Tablet", "value": 4400 }
```

---

## Steps to Run

1. Insert the dataset into your MongoDB collection (`sales`).
2. Copy and paste the `mapFunction` and `reduceFunction` into your MongoDB shell or script.
3. Run the `mapReduce` command.
4. Use `db.sales_summary.find()` to view the summarized output.

---

## 15.

1. Create a Table as **workers** and the details are

| S.No | Name | Designation | Branch |
|---|---|---|---|
| 1 | Ram | Manager | Chennai |
| 2 | Santhosh | Supervisor | Madurai |
| 3 | Hari | Assistant | Trichy |

Perform the following:

- Alter the table by adding a column **Salary**
- Alter the table by modifying the column **Name**
- Describe the table **employee**
- Copy the table **employee** as **emp**
- Truncate the table
- Delete the Second row from the table
- Drop the table

## 1. Create the `workers` table

```
sql
Copy code
CREATE TABLE workers (
    SNo INT PRIMARY KEY,
    Name VARCHAR(50),
    Designation VARCHAR(50),
```

```
    Branch VARCHAR(50)
);

INSERT INTO workers (SNo, Name, Designation, Branch)
VALUES
    (1, 'Ram', 'Manager', 'Chennai'),
    (2, 'Santhosh', 'Supervisor', 'Madurai'),
    (3, 'Hari', 'Assistant', 'Trichy');
```

---

## 2. Alter the table by adding a column `Salary`

```
sql
Copy code
ALTER TABLE workers ADD Salary DECIMAL(10, 2);
```

---

## 3. Alter the table by modifying the column `Name`

```
sql
Copy code
ALTER TABLE workers MODIFY Name VARCHAR(100);
```

---

## 4. Describe the table `workers`

To describe the structure of the table:

```
sql
Copy code
DESC workers;
```

---

## 5. Copy the table `workers` as `emp`

```
sql
Copy code
CREATE TABLE emp AS SELECT * FROM workers;
```

---

## 6. Truncate the table

To remove all data from the table:

```
sql
Copy code
TRUNCATE TABLE workers;
```

---

## 7. Delete the second row from the table

Before truncating, you can delete a specific row:

```
sql
Copy code
DELETE FROM workers WHERE SNo = 2;
```

---

## 8. Drop the table

To permanently remove the table:

```
sql
Copy code
DROP TABLE workers;
```

---

## Steps to Execute:

1. Use the `CREATE TABLE` and `INSERT` statements to create the `workers` table and insert the records.
2. Execute the queries in order using an SQL interface such as MySQL Workbench, phpMyAdmin, or a command-line tool.
3. Verify the results of each step using `SELECT` statements where applicable.