Q1. Create a database and a table to store employee details. Perform basic operations like INSERT, UPDATE, and DELETE using SELECT queries.

```
-- Create the Employee table
CREATE TABLE Employee (
   EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
   Name VARCHAR(100) NOT NULL,
   Position VARCHAR(100),
   Salary DECIMAL(10, 2) CHECK (Salary > 0),
   JoiningDate DATE NOT NULL,
   ActiveStatus BOOLEAN DEFAULT TRUE
);

-- Insert sample employee data
INSERT INTO Employee (Name, Position, Salary, JoiningDate, ActiveStatus)
VALUES
('Mansi Chalak', 'Manager', 75000.00, '2023-01-01', TRUE),
('Janhavi Pawar', 'Developer', 60000.00, '2022-05-10', TRUE),
('Gauri Dani', 'Tester', 50000.00, '2021-08-15', TRUE),
('Sanjivani Mogare', 'Designer', 45000.00, '2020-03-20', FALSE);

-- View all employees
SELECT * FROM Employee;

-- View only active employees
SELECT EmployeeID, Name, Position, Salary FROM Employee WHERE ActiveStatus = TRUE;

-- Increase salary of employee with ID 2 by 10%
UPDATE Employee
```

```
SET Salary = Salary * 1.10
WHERE EmployeeID = 2;

-- Reactivate employee with ID 4
UPDATE Employee
SET ActiveStatus = TRUE
WHERE EmployeeID = 4;

-- Delete employee with ID 3
DELETE FROM Employee WHERE EmployeeID = 3;

-- Show employees who joined after 1st Jan 2021
SELECT * FROM Employee WHERE JoiningDate > '2021-01-01';

-- Show employees with salary greater than 60000
SELECT Name, Salary FROM Employee WHERE Salary > 60000;

-- Show highest and lowest salaries
SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;

-- Show top 3 highest paid employees
SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
```

Q2. Design an ER diagram for a Roadway Travel Management System with entities like Customer, Travel Route, and Booking. Create tables and perform operations such as bookings and route assignments.

```
-- Create Customer Table
```

```
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100),
    PhoneNumber VARCHAR(15),
    Address VARCHAR(255)
);

-- Create Travel Route Table
CREATE TABLE TravelRoute (
    RouteID INT PRIMARY KEY AUTO_INCREMENT,
    StartLocation VARCHAR(100),
    EndLocation VARCHAR(100),
    Distance DECIMAL(5, 2),
    Price DECIMAL(10, 2)
);

-- Create Booking Table
CREATE TABLE Booking (
    BookingID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    RouteID INT,
    BookingDate DATE NOT NULL,
    SeatNumber INT,
    TotalPrice DECIMAL(10, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (RouteID) REFERENCES TravelRoute(RouteID)
);
```

```
-- Insert Sample Data into Customer Table
INSERT INTO Customer (Name, Email, PhoneNumber, Address) VALUES
('Mansi Chalak', 'mansi@example.com', '1112223333', 'Nashik, Maharashtra'),
('Sakshi Udawant', 'sakshi@example.com', '4445556666', 'Pune, Maharashtra');

-- Insert Sample Data into TravelRoute Table
INSERT INTO TravelRoute (StartLocation, EndLocation, Distance, Price) VALUES
('Nashik', 'Pune', 210.00, 300.00),
('Pune', 'Mumbai', 150.00, 250.00),
('Mumbai', 'Nagpur', 700.00, 800.00);

-- Insert Bookings
INSERT INTO Booking (CustomerID, RouteID, BookingDate, SeatNumber, TotalPrice) VALUES
(1, 1, '2025-04-05', 12, 300.00),
(2, 2, '2025-04-06', 14, 250.00);

-- Retrieve All Customers
SELECT * FROM Customer;

-- Retrieve All Travel Routes
SELECT * FROM TravelRoute;

-- Retrieve All Bookings
SELECT * FROM Booking;

-- Retrieve All Bookings for Mansi Chalak (CustomerID = 1)
```

```
SELECT b.BookingID, c.Name AS CustomerName, r.StartLocation, r.EndLocation,
b.BookingDate, b.SeatNumber, b.TotalPrice
FROM Booking b
JOIN Customer c ON b.CustomerID = c.CustomerID
JOIN TravelRoute r ON b.RouteID = r.RouteID
WHERE c.CustomerID = 1;

-- Update Seat Number and Price for Mansi's Booking
UPDATE Booking
SET SeatNumber = 18, TotalPrice = 320.00
WHERE BookingID = 1;

-- Delete Sakshi's Booking
DELETE FROM Booking WHERE BookingID = 2;

-- Show Available Routes Not Yet Booked by Mansi
SELECT r.RouteID, r.StartLocation, r.EndLocation, r.Price
FROM TravelRoute r
WHERE NOT EXISTS (
    SELECT 1
    FROM Booking b
    WHERE b.RouteID = r.RouteID
    AND b.CustomerID = 1
);
```

Q3. Create a table with columns for EmployeeID, Name, Salary, JoiningDate, and ActiveStatus using different data types. Insert sample data and perform queries to manipulate and retrieve data.

```sql
-- Create the Employee Table
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Salary DECIMAL(10,2) CHECK (Salary > 0),
    JoiningDate DATE NOT NULL,
    ActiveStatus BOOLEAN DEFAULT TRUE
);

-- Insert Sample Data with the names you provided
INSERT INTO Employee (Name, Salary, JoiningDate, ActiveStatus)
VALUES
('Mansi Chalak', 55000.00, '2023-06-15', TRUE),
('Janhavi Pawar', 72000.50, '2022-09-25', TRUE),
('Gauri Dani', 48000.75, '2021-12-10', FALSE),
('Sanjivani Mogare', 63000.00, '2020-07-05', TRUE);

-- Retrieve All Employees
SELECT * FROM Employee;

-- Retrieve Active Employees
SELECT EmployeeID, Name, Salary FROM Employee WHERE ActiveStatus = TRUE;

-- Increase Salary of Employee with EmployeeID = 2 (Janhavi Pawar)
UPDATE Employee
SET Salary = Salary * 1.10
WHERE EmployeeID = 2;
```

```
-- Change Active Status of Employee with EmployeeID = 4 (Sanjivani Mogare)
UPDATE Employee
SET ActiveStatus = FALSE
WHERE EmployeeID = 4;

-- Delete Employee Record with EmployeeID = 3 (Gauri Dani)
DELETE FROM Employee
WHERE EmployeeID = 3;

-- Retrieve Employees Who Joined in 2023
SELECT * FROM Employee
WHERE YEAR(JoiningDate) = 2023;

-- Retrieve Employees with Salary Greater Than 60,000
SELECT Name, Salary FROM Employee
WHERE Salary > 60000;

-- Find the Highest and Lowest Salary in the Organization
SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary
FROM Employee;

-- Retrieve the Top 3 Highest Paid Employees
SELECT * FROM Employee
ORDER BY Salary DESC
LIMIT 3;
```

Q4. Create a table to store employee information with constraints like Primary Key, Foreign Key, and Unique.

```
CREATE TABLE Department (
   DeptID INT PRIMARY KEY,
   DeptName VARCHAR(50) UNIQUE
);

CREATE TABLE Employee (
   EmpID INT PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
   Email VARCHAR(100) UNIQUE,
   Salary DECIMAL(10,2) CHECK (Salary > 0),
   DeptID INT REFERENCES Department(DeptID)
);

INSERT INTO Department (DeptID, DeptName) VALUES (1, 'HR');
INSERT INTO Department (DeptID, DeptName) VALUES (2, 'IT');

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (101, 'Mansi Chalak', 'mansi@example.com', 50000.00, 1);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (102, 'Swara Jadhav', 'swara@example.com', 60000.00, 2);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (103, 'Janhavi Pawar', 'janhavi@example.com', 55000.00, 1);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
```

```
VALUES (104, 'Sakshi Udawant', 'sakshi@example.com', 45000.00, 2);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (105, 'Sakshi Udawant', 'sakshi2@example.com', 40000.00, 1);

SELECT * FROM Department;
SELECT * FROM Employee;
```

Q5. To Test constraints like PRIMARY KEY, UNIQUE, and CHECK by inserting invalid data into the Employee table.

```
CREATE TABLE Customer (
   CustomerID INT PRIMARY KEY,
   FirstName VARCHAR(100) NOT NULL,
   LastName VARCHAR(100) NOT NULL,
   Email VARCHAR(100) UNIQUE,
   Phone VARCHAR(15),
   Age INT CHECK (Age >= 18),
   IsActive BOOLEAN DEFAULT TRUE
);

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age, IsActive)
VALUES (1, 'Mansi', 'Chalak', 'mansi.chalak@example.com', '1234567890', 25, TRUE);

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
VALUES (2, 'Swara', 'Jadhav', 'swara.jadhav@example.com', '0987654321', 30);

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
VALUES (3, 'NULL', 'Udawant', 'sakshi.udawant@example.com', '5551234567', 20);
```

```
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
VALUES (4, 'Rachana', 'Shinde', 'rachana.shinde@example.com', '6669876543', 18);

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
VALUES (5, 'Dimpal', 'Tile', 'dimpal.tile@example.com', '7771234567', 28);

SELECT * FROM Customer;
```

Q6. Use DDL commands to create tables and DML commands to insert, update, and delete data. Write SELECT queries to retrieve and verify data changes.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT,
    Department VARCHAR(50),
    Salary DECIMAL(10, 2)
);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)
VALUES (1, 'Mansi', 'Chalak', 28, 'HR', 50000.00);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)
VALUES (2, 'Swara', 'Jadhav', 35, 'IT', 65000.00);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)
VALUES (3, 'Rachana', 'Shinde', 40, 'Finance', 75000.00);
```

```
UPDATE Employees
SET Salary = 70000.00
WHERE EmployeeID = 2;

UPDATE Employees
SET FirstName = 'Swara', LastName = 'Jadhav', Salary = 75000.00
WHERE EmployeeID = 2;

UPDATE Employees
SET FirstName = 'Rachana', LastName = 'Shinde', Age = 45, Department = 'Management',
Salary = 80000.00
WHERE EmployeeID = 3;

UPDATE Employees
SET Salary = Salary * 1.10
WHERE Department = 'HR';

UPDATE Employees
JOIN (SELECT MAX(Salary) AS MaxSalary FROM Employees) AS Sub
SET Employees.Salary = Employees.Salary + 5000
WHERE Employees.Salary = Sub.MaxSalary;

UPDATE Employees
SET Salary = CASE
    WHEN Department = 'HR' THEN Salary * 1.05
    WHEN Department = 'IT' THEN Salary * 1.08
```

```
    WHEN Department = 'Finance' THEN Salary * 1.10
    ELSE Salary
END;

DELETE FROM Employees
WHERE EmployeeID = 1;

SELECT * FROM Employees;

SELECT * FROM Employees
WHERE EmployeeID = 2;

SELECT * FROM Employees
WHERE EmployeeID = 1;
```

Q7. Create a Sales table and use aggregate functions like COUNT, SUM, AVG, MIN, and MAX to summarize sales data and calculate statistics.

```
-- Create the Sales Table
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY AUTO_INCREMENT,
    Product VARCHAR(50),
    Quantity INT,
    Price DECIMAL(10,2),
    SaleDate DATE
);

-- Insert Sample Data into Sales Table
INSERT INTO Sales (Product, Quantity, Price, SaleDate) VALUES
```

```
('Laptop', 2, 75000.00, '2025-02-01'),
('Mobile', 5, 20000.00, '2025-02-02'),
('Tablet', 3, 30000.00, '2025-02-03'),
('Laptop', 1, 78000.00, '2025-02-04'),
('Mobile', 4, 22000.00, '2025-02-05'),
('Tablet', 2, 32000.00, '2025-02-06');

-- Count the Total Number of Sales Records
SELECT COUNT(*) AS Total_Sales FROM Sales;

-- Sum of Total Revenue Generated
SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;

-- Average Price of Products Sold
SELECT AVG(Price) AS Average_Price FROM Sales;

-- Minimum and Maximum Price of a Product Sold
SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;

-- Count the Total Number of Sales Records
SELECT COUNT(*) AS Total_Sales FROM Sales;

-- Count the Number of Distinct Products Sold
SELECT COUNT(DISTINCT Product) AS Unique_Products FROM Sales;

-- Count the Number of Sales Per Product
SELECT Product, COUNT(*) AS Sales_Count
FROM Sales
```

```
GROUP BY Product;

-- Count the Number of Sales Per Day
SELECT SaleDate, COUNT(*) AS Sales_Per_Day
FROM Sales
GROUP BY SaleDate;

-- Count the Number of Sales Where More Than 2 Units Were Sold
SELECT COUNT(*) AS High_Quantity_Sales
FROM Sales
WHERE Quantity > 2;

-- Count the Number of Sales in the Current Month
SELECT COUNT(*) AS Sales_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

-- Count the Number of Sales Transactions Where Total Sale Value Was More Than ₹50,000
SELECT COUNT(*) AS High_Value_Sales
FROM Sales
WHERE (Quantity * Price) > 50000;

-- Count the Number of Sales Records for Each Product Where Total Sale Value Is Greater Than ₹40,000
SELECT Product, COUNT(*) AS High_Value_Transactions
FROM Sales
WHERE (Quantity * Price) > 40000
```

```
GROUP BY Product;

-- Count the Number of Sales Made After a Specific Date (e.g., Feb 3, 2025)
SELECT COUNT(*) AS Sales_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';

-- Sum of Total Revenue Generated
SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;

-- Sum of Total Quantity of Products Sold
SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;

-- Sum of Total Revenue Per Product
SELECT Product, SUM(Quantity * Price) AS Revenue_Per_Product
FROM Sales
GROUP BY Product;

-- Sum of Total Revenue Per Day
SELECT SaleDate, SUM(Quantity * Price) AS Revenue_Per_Day
FROM Sales
GROUP BY SaleDate;

-- Sum of Total Revenue in the Current Month
SELECT SUM(Quantity * Price) AS Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
```

```
-- Sum of Revenue for Sales Where Quantity Sold Is Greater Than 2
SELECT SUM(Quantity * Price) AS High_Quantity_Revenue
FROM Sales
WHERE Quantity > 2;

-- Sum of Total Revenue Generated After a Specific Date (e.g., Feb 3, 2025)
SELECT SUM(Quantity * Price) AS Revenue_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';

-- Sum of Revenue Per Product Where the Total Revenue Per Transaction Is Greater Than ₹40,000
SELECT Product, SUM(Quantity * Price) AS High_Value_Revenue
FROM Sales
WHERE (Quantity * Price) > 40000
GROUP BY Product;

-- Average Price of Products Sold
SELECT AVG(Price) AS Average_Price FROM Sales;

-- Average Quantity of Products Sold Per Transaction
SELECT AVG(Quantity) AS Average_Quantity_Sold FROM Sales;

-- Average Revenue Per Transaction
SELECT AVG(Quantity * Price) AS Average_Revenue_Per_Transaction
FROM Sales;
```

```
-- Average Price Per Product
SELECT Product, AVG(Price) AS Average_Price_Per_Product
FROM Sales
GROUP BY Product;

-- Average Revenue Per Product
SELECT Product, AVG(Quantity * Price) AS Average_Revenue_Per_Product
FROM Sales
GROUP BY Product;

-- Average Quantity Sold Per Product
SELECT Product, AVG(Quantity) AS Average_Quantity_Per_Product
FROM Sales
GROUP BY Product;

-- Average Revenue Per Day
SELECT SaleDate, AVG(Quantity * Price) AS Average_Revenue_Per_Day
FROM Sales
GROUP BY SaleDate;

-- Average Revenue in the Current Month
SELECT AVG(Quantity * Price) AS Average_Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

-- Average Price of Products Where More Than 2 Units Were Sold
SELECT AVG(Price) AS Avg_Price_High_Quantity_Sales
```

```
FROM Sales

WHERE Quantity > 2;


-- Average Revenue After a Specific Date (e.g., Feb 3, 2025)

SELECT AVG(Quantity * Price) AS Average_Revenue_After_Date

FROM Sales

WHERE SaleDate > '2025-02-03';


-- Minimum and Maximum Price of a Product Sold

SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;


-- Minimum and Maximum Quantity of Products Sold in a Single Transaction

SELECT MIN(Quantity) AS Min_Quantity_Sold, MAX(Quantity) AS Max_Quantity_Sold FROM
Sales;


-- Minimum and Maximum Revenue Generated from a Single Transaction

SELECT MIN(Quantity * Price) AS Min_Revenue, MAX(Quantity * Price) AS Max_Revenue
FROM Sales;


-- Minimum and Maximum Price Per Product

SELECT Product, MIN(Price) AS Min_Price_Per_Product, MAX(Price) AS
Max_Price_Per_Product

FROM Sales

GROUP BY Product;


-- Minimum and Maximum Revenue Per Product

SELECT Product, MIN(Quantity * Price) AS Min_Revenue_Per_Product, MAX(Quantity *
Price) AS Max_Revenue_Per_Product

FROM Sales
```

```
GROUP BY Product;


-- Minimum and Maximum Quantity Sold Per Product

SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity) AS
Max_Quantity_Per_Product

FROM Sales

GROUP BY Product;


-- Minimum and Maximum Revenue Per Day

SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quantity * Price)
AS Max_Revenue_Per_Day

FROM Sales

GROUP BY SaleDate;


-- Minimum and Maximum Revenue in the Current Month

SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity * Price) AS
Max_Revenue_This_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT_DATE);


-- Minimum and Maximum Price of Products Where More Than 2 Units Were Sold

SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS
Max_Price_High_Quantity_Sales

FROM Sales

WHERE Quantity > 2;


-- Minimum and Maximum Revenue After a Specific Date (e.g., Feb 3, 2025)
```

```
SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity * Price) AS
Max_Revenue_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';
```

Q8. Given Customers and Orders tables, write SQL queries to perform INNER JOIN, LEFT JOIN, and RIGHT JOIN to retrieve combined data for customer orders.

```
-- Create the Customers Table
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100) NOT NULL
);

-- Create the Orders Table
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    order_date DATE NOT NULL,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);

-- Insert Customer Data with the given customer names and IDs
INSERT INTO Customers (customer_id, customer_name) VALUES
(1075, 'Mansi Chalak'),
(1074, 'Rachana Shinde'),
(1099, 'Dimpal Tile'),
(1103, 'Samruddhi Gunjal');
```

```
-- Insert some example order data linked to the customers
INSERT INTO Orders (order_id, order_date, customer_id) VALUES
(101, '2024-01-01', 1075),
(102, '2024-01-02', 1074),
(103, '2024-01-03', 1099),
(104, '2024-02-10', 1103);

-- Select all customers
SELECT * FROM Customers;

-- Select all orders
SELECT * FROM Orders;

-- INNER JOIN: Get customers with their corresponding orders
SELECT
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date
FROM
    Customers c
INNER JOIN
    Orders o
ON
    c.customer_id = o.customer_id;

-- LEFT JOIN: Get all customers with their orders (if any)
SELECT
```

```
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date
FROM
    Customers c
LEFT JOIN
    Orders o
ON
    c.customer_id = o.customer_id;

-- RIGHT JOIN: Get all orders with their corresponding customer details (if any)
SELECT
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date
FROM
    Customers c
RIGHT JOIN
    Orders o
ON
    c.customer_id = o.customer_id;
```