1 Install and set up MySQL. Create a database and a table to store employee details. Perform basic operations like INSERT & DELETE.

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Age INT,

Department VARCHAR(50)

);

INSERT INTO Employees (Name, Age, Department) VALUES ('John Doe', 30, 'IT');

INSERT INTO Employees (Name, Age, Department) VALUES ('Jane Smith', 25, 'HR');

DELETE FROM Employees WHERE EmployeeID = 1;

SELECT \* FROM Employees;

2 Create a table for storing student information. Insert sample data and perform basic operations: INSERT, UPDATE, DELETE, and SELECT.

1 CREATE DATABASE StudentDB;

USE StudentDB;

2 CREATE TABLE Students (

StudentID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Age INT,

Grade VARCHAR(10),

City VARCHAR(50)

);

Step 3: Insert Sample Data

3 INSERT INTO Students (Name, Age, Grade, City) VALUES ('Alice Johnson', 18, '12th', 'Mumbai');

INSERT INTO Students (Name, Age, Grade, City) VALUES ('Bob Smith', 17, '11th', 'Delhi');

INSERT INTO Students (Name, Age, Grade, City) VALUES ('Charlie Brown', 19, '12th', 'Bangalore');

Step 4: Update a Record

UPDATE Students

SET City = 'Pune'

WHERE Name = 'Alice Johnson';

tep 5: Delete a Record

SELECT \* FROM Students;

3 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.

**Step 1: Create the Table**

sql

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT, -- Integer (Auto Increment)

Name VARCHAR(100) NOT NULL, -- String (Up to 100 characters)

Salary DECIMAL(10,2), -- Decimal (10 digits, 2 decimal places)

JoiningDate DATE, -- Date

ActiveStatus BOOLEAN -- Boolean (True/False)

);

**tep 2: Insert Sample Data**

sql

INSERT INTO Employees (Name, Salary, JoiningDate, ActiveStatus)

VALUES ('Alice Johnson', 55000.00, '2022-06-15', TRUE);

INSERT INTO Employees (Name, Salary, JoiningDate, ActiveStatus)

VALUES ('Bob Smith', 62000.50, '2021-08-20', FALSE);

INSERT INTO Employees (Name, Salary, JoiningDate, ActiveStatus)

VALUES ('Charlie Brown', 70000.75, '2020-05-10', TRUE);

**Step 3: Update Employee Details**

Let's say we want to update Charlie’s salary:

sql

UPDATE Employees

SET Salary = 75000.00

WHERE Name = 'Charlie Brown';

**Step 4: Delete an Employee Record**

If Bob Smith has left the company:

sql

DELETE FROM Employees WHERE Name = 'Bob Smith';

**Step 5: Retrieve Employee Data**

* To get all employees:

sql

SELECT \* FROM Employees;

* To fetch only active employees:

sql

SELECT \* FROM Employees WHERE ActiveStatus = TRUE;

* To get employees who joined before 2022:

sql

SELECT \* FROM Employees WHERE JoiningDate < '2022-01-01';

4 .Create a table to store employee information with constraints like Primary Key, Foreign Key, and Unique. Insert valid and invalid data to test the constraints.

**Step 1: Create Departments Table (Reference Table)**

sql

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY AUTO\_INCREMENT, -- Primary Key (Auto Increment)

DepartmentName VARCHAR(50) UNIQUE -- Unique Constraint (No duplicate names)

);

**Step 2: Create Employees Table**

sql

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT, -- Primary Key (Auto Increment)

Name VARCHAR(100) NOT NULL, -- Not Null (Name cannot be empty)

Email VARCHAR(100) UNIQUE, -- Unique Constraint (No duplicate emails)

Salary DECIMAL(10,2) CHECK (Salary > 0), -- Check Constraint (Salary must be positive)

JoiningDate DATE, -- Date Format

DepartmentID INT, -- Foreign Key Reference

ActiveStatus BOOLEAN,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID) -- Foreign Key Constraint

);

**Step 3: Insert Valid Data**

sql

INSERT INTO Departments (DepartmentName) VALUES ('IT'), ('HR'), ('Finance');

INSERT INTO Employees (Name, Email, Salary, JoiningDate, DepartmentID, ActiveStatus)

VALUES ('Alice Johnson', 'alice@example.com', 55000.00, '2022-06-15', 1, TRUE);

INSERT INTO Employees (Name, Email, Salary, JoiningDate, DepartmentID, ActiveStatus)

VALUES ('Bob Smith', 'bob@example.com', 62000.50, '2021-08-20', 2, FALSE);

**Step 4: Insert Invalid Data (To Test Constraints)**

sql

-- Attempting to insert duplicate email (Fails due to UNIQUE constraint)

INSERT INTO Employees (Name, Email, Salary, JoiningDate, DepartmentID, ActiveStatus)

VALUES ('Charlie Brown', 'alice@example.com', 70000.75, '2020-05-10', 1, TRUE);

-- Attempting to insert negative salary (Fails due to CHECK constraint)

INSERT INTO Employees (Name, Email, Salary, JoiningDate, DepartmentID, ActiveStatus)

VALUES ('Daniel Green', 'daniel@example.com', -50000, '2022-03-01', 3, TRUE);

-- Attempting to insert invalid DepartmentID (Fails due to FOREIGN KEY constraint)

INSERT INTO Employees (Name, Email, Salary, JoiningDate, DepartmentID, ActiveStatus)

VALUES ('Emma Watson', 'emma@example.com', 60000, '2023-01-12', 5, TRUE);

**Step 5: Query Data**

* To retrieve all employees:

sql

SELECT \* FROM Employees;

* To fetch employees who belong to a specific department:

sql

SELECT Employees.Name, Employees.Salary, Departments.DepartmentName

FROM Employees

JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

5 Create a table for Customer details with various integrity constraints like NOT NULL, CHECK, and DEFAULT. Insert valid and invalid data to test these constraints and ensure data integrity.

**Step 1: Create Customer Table with Constraints**

sql

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT, -- Primary Key (Auto Increment)

Name VARCHAR(100) NOT NULL, -- NOT NULL Constraint (Name cannot be empty)

Age INT CHECK (Age >= 18), -- CHECK Constraint (Age must be 18+)

Email VARCHAR(100) UNIQUE, -- UNIQUE Constraint (No duplicate emails)

City VARCHAR(50) DEFAULT 'Unknown' -- DEFAULT Constraint (If City not provided)

);

**Step 2: Insert Valid Data**

sql

INSERT INTO Customers (Name, Age, Email, City) VALUES ('Alice Johnson', 25, 'alice@example.com', 'Mumbai');

INSERT INTO Customers (Name, Age, Email, City) VALUES ('Bob Smith', 30, 'bob@example.com', 'Delhi');

**Step 3: Insert Invalid Data (Fails Constraints)**

sql

-- Fails because Age is less than 18 (CHECK Constraint)

INSERT INTO Customers (Name, Age, Email, City) VALUES ('Charlie Brown', 16, 'charlie@example.com', 'Bangalore');

-- Fails because Email is duplicated (UNIQUE Constraint)

INSERT INTO Customers (Name, Age, Email, City) VALUES ('David Green', 27, 'alice@example.com', 'Pune');

-- City defaults to 'Unknown' when not provided (DEFAULT Constraint)

INSERT INTO Customers (Name, Age, Email) VALUES ('Emily Watson', 22, 'emily@example.com');

**Step 4: Retrieve Data**

sql

SELECT \* FROM Customers;

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

**Step 1: Use DDL Commands to Create Tables**

DDL commands define the structure of the database.

**Create a Customers Table**

sql

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Age INT CHECK (Age >= 18),

Email VARCHAR(100) UNIQUE,

City VARCHAR(50) DEFAULT 'Unknown'

);

**Create an Orders Table**

sql

CREATE TABLE Orders (

OrderID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT,

OrderAmount DECIMAL(10,2),

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Step 2: Use DML Commands to Insert, Update, and Delete Data**

DML commands manipulate data within the tables.

**Insert Data into Customers Table**

sql

INSERT INTO Customers (Name, Age, Email, City)

VALUES ('Alice Johnson', 25, 'alice@example.com', 'Mumbai');

INSERT INTO Customers (Name, Age, Email, City)

VALUES ('Bob Smith', 30, 'bob@example.com', 'Delhi');

**Insert Data into Orders Table**

sql

INSERT INTO Orders (CustomerID, OrderAmount, OrderDate)

VALUES (1, 500.50, '2025-04-10');

INSERT INTO Orders (CustomerID, OrderAmount, OrderDate)

VALUES (2, 750.75, '2025-04-09');

**Update Data**

Let's update **Alice's city**:

sql

UPDATE Customers

SET City = 'Pune'

WHERE Name = 'Alice Johnson';

**Delete Data**

Let's delete **Bob’s order**:

sql

DELETE FROM Orders WHERE CustomerID = 2;

**Step 3: Use SELECT Queries to Retrieve Data**

Use SELECT statements to check and verify changes.

**Retrieve All Customers**

sql

SELECT \* FROM Customers;

**Retrieve Orders Details**

sql

SELECT \* FROM Orders;

**Find Customers with Orders**

sql

SELECT Customers.Name, Orders.OrderAmount, Orders.OrderDate

FROM Customers

JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

7 Create a Sales table and use aggregate functions like COUNT, SUM, AVG, MIN, and MAXtosummarize sales data and calculate statistics.

**Step 1: Create the Sales Table**

sql

CREATE TABLE Sales (

SaleID INT PRIMARY KEY AUTO\_INCREMENT, -- Primary Key

Product VARCHAR(100), -- Product Name

Quantity INT CHECK (Quantity > 0), -- Quantity (must be greater than 0)

Price DECIMAL(10,2), -- Price per unit

SaleDate DATE -- Date of sale

);

**Step 2: Insert Sample Sales Data**

sql

INSERT INTO Sales (Product, Quantity, Price, SaleDate) VALUES ('Laptop', 2, 50000, '2025-04-01');

INSERT INTO Sales (Product, Quantity, Price, SaleDate) VALUES ('Mobile', 5, 20000, '2025-04-02');

INSERT INTO Sales (Product, Quantity, Price, SaleDate) VALUES ('Tablet', 3, 15000, '2025-04-03');

INSERT INTO Sales (Product, Quantity, Price, SaleDate) VALUES ('Headphones', 4, 5000, '2025-04-04');

**Step 3: Use Aggregate Functions to Summarize Sales Data**

**1. Count Total Number of Sales**

sql

SELECT COUNT(\*) AS TotalSales FROM Sales;

**2. Calculate Total Revenue**

sql

SELECT SUM(Quantity \* Price) AS TotalRevenue FROM Sales;

**3. Find the Average Price of Products**

sql

SELECT AVG(Price) AS AveragePrice FROM Sales;

**4. Find the Minimum and Maximum Sale Prices**

sql

SELECT MIN(Price) AS MinPrice, MAX(Price) AS MaxPrice FROM Sales;

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

**Step 1: Create Customers and Orders Tables**

sql

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Email VARCHAR(100) UNIQUE

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT,

OrderAmount DECIMAL(10,2),

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Step 2: Insert Sample Data**

sql

INSERT INTO Customers (Name, Email) VALUES ('Alice Johnson', 'alice@example.com');

INSERT INTO Customers (Name, Email) VALUES ('Bob Smith', 'bob@example.com');

INSERT INTO Customers (Name, Email) VALUES ('Charlie Brown', 'charlie@example.com');

INSERT INTO Orders (CustomerID, OrderAmount, OrderDate) VALUES (1, 500.50, '2025-04-10');

INSERT INTO Orders (CustomerID, OrderAmount, OrderDate) VALUES (2, 750.75, '2025-04-09');

**Step 3: Perform Different Types of JOINs**

**1. INNER JOIN (Returns matching records in both tables)**

sql

SELECT Customers.Name, Orders.OrderAmount, Orders.OrderDate

FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

💡 **This will return only customers who have placed orders.**

**2. LEFT JOIN (Returns all customers, even those without orders)**

sql

SELECT Customers.Name, Orders.OrderAmount, Orders.OrderDate

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

💡 **This ensures that all customers are listed, even if they have not placed orders.**

**3. RIGHT JOIN (Returns all orders, even if customer data is missing)**

sql

SELECT Customers.Name, Orders.OrderAmount, Orders.OrderDate

FROM Customers

RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;