ASSIGNMENT 1
Identify primary keys and foreign keys for following database. Create tables and execute queries for given statements.
employee(eid,ename,salary)
assignment(projectid,eid)
project(projectid,project_name,manager)
manager(eid,ename)
Write queries for the following questions:
1. Alter table to add address in employee table.
2. Display employee name and projects on which they are working/
3. Display projectid, projectname and their managers.
4. Create view of employees working on 'Bank Management' project.
5. Print names of employees whose salary is greater than 40000
6. Update salary of each employee with increase of Rs.2000

ANSWERS

```
DROP DATABASE IF EXISTS company_db;
CREATE DATABASE company_db;
USE company_db;

DROP TABLE IF EXISTS assignment;
DROP TABLE IF EXISTS project;
DROP TABLE IF EXISTS manager;
DROP TABLE IF EXISTS employee;


CREATE TABLE employee (
   eid INT PRIMARY KEY,
   ename VARCHAR(50),
   salary INT
);

CREATE TABLE manager (
   eid INT PRIMARY KEY,
   ename VARCHAR(50)
);

CREATE TABLE project (
   projectid INT PRIMARY KEY,
   project_name VARCHAR(100),
   manager INT,
   FOREIGN KEY (manager) REFERENCES manager(eid)
);

CREATE TABLE assignment (
   projectid INT,
   eid INT,
   PRIMARY KEY (projectid, eid),
   FOREIGN KEY (projectid) REFERENCES project(projectid),
   FOREIGN KEY (eid) REFERENCES employee(eid)
);
-- employee
INSERT INTO employee VALUES (1, 'Satyam', 45000);
INSERT INTO employee VALUES (2, 'Shashank', 39000);
INSERT INTO employee VALUES (3, 'Saurav', 42000);
INSERT INTO employee VALUES (4, 'Adarsh', 38000);
```

```
INSERT INTO employee VALUES (5, 'Shivam', 41000);

OR

INSERT INTO employee VALUES
(1, 'Satyam', 45000),
(2, 'Shashank', 39000),
(3, 'Saurav', 42000),
(4, 'Adarsh', 38000),
(5, 'Shivam', 41000);


-- manager
INSERT INTO manager VALUES (1, 'Satyam');
INSERT INTO manager VALUES (2, 'Shashank');

OR
INSERT INTO manager VALUES
(1, 'Satyam'),
(2, 'Shashank');

-- project
INSERT INTO project VALUES (101, 'Bank Management', 1);
INSERT INTO project VALUES (102, 'Hospital Portal', 2);

OR

INSERT INTO project VALUES
(101, 'Bank Management', 1),
(102, 'Hospital Portal', 2);

-- assignment
INSERT INTO assignment VALUES (101, 1);
INSERT INTO assignment VALUES (101, 3);
INSERT INTO assignment VALUES (102, 2);
INSERT INTO assignment VALUES (102, 4);

OR
INSERT INTO assignment VALUES
(101, 1),
(101, 3),
(102, 2),
(102, 4);



ALTER TABLE employee ADD address VARCHAR(100);

SELECT e.ename, p.project_name
FROM employee e
JOIN assignment a ON e.eid = a.eid
JOIN project p ON a.projectid = p.projectid;

SELECT p.projectid, p.project_name, m.ename AS manager_name
FROM project p
JOIN manager m ON p.manager = m.eid;
```

```
CREATE VIEW bank_project_employees AS
SELECT e.eid, e.ename
FROM employee e
JOIN assignment a ON e.eid = a.eid
JOIN project p ON a.projectid = p.projectid
WHERE p.project_name = 'Bank Management';

SELECT ename FROM employee
WHERE salary > 40000;

UPDATE employee
SET salary = salary + 2000;

EXTRA for showing updated salaries

SELECT eid, ename, salary
FROM employee;
```

ASSIGNMENT 2

Identify primary keys and foreign keys for following database. Create tables and execute queries for given statements.
employee(eid, ename, salary)
assignment(projectid,eid)
project(projectid,project_name,manager)
manager(eid,ename)
Write queries for the following questions:
1. Modify eid to use auto_increment
2. Display Employees working in both projects 'Bank Management' and 'Content Management'. 3. Display average salary of organization.
4. Display employees who do not work on 'Bank Management' Project.
5. Delete employee whose id is 5.
6. Display employee having highest salary in oraganization.

ANSWERS

```
DROP DATABASE IF EXISTS organization_db;
CREATE DATABASE organization_db;
USE organization_db;

DROP TABLE IF EXISTS assignment;
DROP TABLE IF EXISTS project;
DROP TABLE IF EXISTS manager;
DROP TABLE IF EXISTS employee;

CREATE TABLE employee (
   eid INT AUTO_INCREMENT PRIMARY KEY,
   ename VARCHAR(50),
   salary INT
);

CREATE TABLE manager (
   eid INT PRIMARY KEY,
   ename VARCHAR(50)
);

CREATE TABLE project (
```

```
    projectid INT PRIMARY KEY,
    project_name VARCHAR(100),
    manager INT,
    FOREIGN KEY (manager) REFERENCES manager(eid)
);

CREATE TABLE assignment (
    projectid INT,
    eid INT,
    PRIMARY KEY (projectid, eid),
    FOREIGN KEY (projectid) REFERENCES project(projectid),
    FOREIGN KEY (eid) REFERENCES employee(eid)
);

INSERT INTO employee (ename, salary) VALUES
('Satyam', 45000),
('Shashank', 39000),
('Saurav', 42000),
('Adarsh', 38000),
('Shivam', 41000);

INSERT INTO manager VALUES
(1, 'Satyam'),
(2, 'Shashank');

INSERT INTO project VALUES
(101, 'Bank Management', 1),
(102, 'Hospital Portal', 2),
(103, 'Content Management', 2);

INSERT INTO assignment VALUES
(101, 1),
(101, 3),
(102, 2),
(102, 4),
(103, 1),
(103, 4);


SELECT e.ename
FROM employee e
JOIN assignment a ON e.eid = a.eid
JOIN project p ON a.projectid = p.projectid
WHERE p.project_name IN ('Bank Management', 'Content Management')
GROUP BY e.eid
HAVING COUNT(DISTINCT p.project_name) = 2;

SELECT AVG(salary) AS avg_salary FROM employee;

SELECT ename
FROM employee
WHERE eid NOT IN (
    SELECT eid
    FROM assignment a
    JOIN project p ON a.projectid = p.projectid
    WHERE p.project_name = 'Bank Management'
);
```

```
DELETE FROM assignment WHERE eid = 5;
DELETE FROM employee WHERE eid = 5;

SELECT ename, salary
FROM employee
WHERE salary = (SELECT MAX(salary) FROM employee);
```

EXTRAS not asked

```
SELECT * FROM employee;
```

ASSIGNMENT 3

Identify primary keys and foreign keys for following database. Create tables and execute queries for given statements.
supplier(supplierid,sname,saddress)
parts(part_id,part_name,color);
catalog(supplierid,part_id,cost);
Write queries for the following questions:
1. Find name of supplier who supply 'green' parts.
2. find name of suppliers who supply both blue and green parts.
3. Find supplier who supply all parts.
4. Find total cost of red parts.
5. Find supplier who supply green parts with minimum cost.
6. Update color of part having part_id = 4 and supplier_id = 2.

ANSWERS

```
DROP DATABASE IF EXISTS supply_db;
CREATE DATABASE supply_db;
USE supply_db;

DROP TABLE IF EXISTS catalog;
DROP TABLE IF EXISTS parts;
DROP TABLE IF EXISTS supplier;

CREATE TABLE supplier (
   supplierid INT PRIMARY KEY,
   sname VARCHAR(50),
   saddress VARCHAR(100)
);

CREATE TABLE parts (
   part_id INT PRIMARY KEY,
   part_name VARCHAR(50),
   color VARCHAR(20)
);

CREATE TABLE catalog (
   supplierid INT,
   part_id INT,
   cost INT,
   PRIMARY KEY (supplierid, part_id),
   FOREIGN KEY (supplierid) REFERENCES supplier(supplierid),
```

```
   FOREIGN KEY (part_id) REFERENCES parts(part_id)
);

INSERT INTO supplier VALUES
(1, 'Satyam', 'Pune'),
(2, 'Adarsh', 'Mumbai'),
(3, 'Saurav', 'Delhi');

INSERT INTO parts VALUES
(1, 'Bolt', 'green'),
(2, 'Nut', 'blue'),
(3, 'Screw', 'red'),
(4, 'Washer', 'green');

INSERT INTO catalog VALUES
(1, 1, 50),
(1, 2, 40),
(1, 3, 60),
(2, 2, 30),
(2, 4, 20),
(3, 1, 45),
(3, 2, 35),
(3, 3, 55),
(3, 4, 25);

SELECT DISTINCT s.sname
FROM supplier s
JOIN catalog c ON s.supplierid = c.supplierid
JOIN parts p ON c.part_id = p.part_id
WHERE p.color = 'green';

SELECT s.sname
FROM supplier s
JOIN catalog c ON s.supplierid = c.supplierid
JOIN parts p ON c.part_id = p.part_id
WHERE p.color IN ('blue', 'green')
GROUP BY s.supplierid
HAVING COUNT(DISTINCT p.color) = 2;

SELECT s.sname
FROM supplier s
JOIN catalog c ON s.supplierid = c.supplierid
GROUP BY s.supplierid
HAVING COUNT(DISTINCT c.part_id) = (SELECT COUNT(*) FROM parts);

SELECT SUM(c.cost) AS total_red_cost
FROM catalog c
JOIN parts p ON c.part_id = p.part_id
WHERE p.color = 'red';

SELECT s.sname, c.cost
FROM supplier s
JOIN catalog c ON s.supplierid = c.supplierid
JOIN parts p ON c.part_id = p.part_id
WHERE p.color = 'green' AND c.cost = (
   SELECT MIN(c2.cost)
   FROM catalog c2
```

```
    JOIN parts p2 ON c2.part_id = p2.part_id
    WHERE p2.color = 'green'
);

UPDATE parts
SET color = 'blue'
WHERE part_id = 4;
```

EXTRAS not asked

```
SELECT * FROM parts;
```

ASSIGNMENT 4

Identify primary keys and foreign keys for following database. Create tables and execute queries for given statements.
emp(eid,ename,street,city);
works(eid,company_name,salary);
company(company_name,city);
manages(eid,manager_id);
Write queries for the following questions:
1. Update company of employee name = ‘Prashant’ from ‘Infosys’ to ‘TCS’. 2. Display names & cities of all employees who work for 'Infosys'
3. Display names & Street address & of all employees who work in TCS cities and earn more than 20000.
4. Find all employees in database who do not work for 'Infosys'.
5. Find company wise total salary.
6. Find names of all employees who work for 'Accenture'.

ANSWERS

```
-- Create Database
CREATE DATABASE company_db;
USE company_db;

-- Create Tables
CREATE TABLE emp (
    eid INT PRIMARY KEY,
    ename VARCHAR(50),
    street VARCHAR(100),
    city VARCHAR(50)
);

CREATE TABLE company (
    company_name VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50)
);

CREATE TABLE works (
    eid INT,
    company_name VARCHAR(50),
    salary INT,
    PRIMARY KEY (eid, company_name),
    FOREIGN KEY (eid) REFERENCES emp(eid),
    FOREIGN KEY (company_name) REFERENCES company(company_name)
);
```

```
CREATE TABLE manages (
   eid INT,
   manager_id INT,
   PRIMARY KEY (eid, manager_id),
   FOREIGN KEY (eid) REFERENCES emp(eid),
   FOREIGN KEY (manager_id) REFERENCES emp(eid)
);

-- Insert Data into emp table
INSERT INTO emp VALUES
(1, 'Prashant', 'Street 1', 'Mumbai'),
(2, 'Amit', 'Street 2', 'Delhi'),
(3, 'Shivam', 'Street 3', 'Bangalore'),
(4, 'Adarsh', 'Street 4', 'Chennai'),
(5, 'Shashank', 'Street 5', 'Pune');

-- Insert Data into company table
INSERT INTO company VALUES
('Infosys', 'Bangalore'),
('TCS', 'Mumbai'),
('Accenture', 'Pune');

-- Insert Data into works table
INSERT INTO works VALUES
(1, 'Infosys', 25000),
(2, 'Infosys', 30000),
(3, 'TCS', 22000),
(4, 'TCS', 18000),
(5, 'Accenture', 35000);

-- Insert Data into manages table
INSERT INTO manages VALUES
(1, 3),
(2, 1),
(3, 5);

UPDATE works
SET company_name = 'TCS'
WHERE eid = (SELECT eid FROM emp WHERE ename = 'Prashant') AND company_name = 'Infosys';

SELECT e.ename, e.city
FROM emp e
JOIN works w ON e.eid = w.eid
WHERE w.company_name = 'Infosys';

SELECT e.ename, e.street, e.city
FROM emp e
JOIN works w ON e.eid = w.eid
JOIN company c ON w.company_name = c.company_name
WHERE c.city = 'Mumbai' AND w.salary > 20000;

SELECT e.ename
FROM emp e
WHERE e.eid NOT IN (SELECT eid FROM works WHERE company_name = 'Infosys');

SELECT w.company_name, SUM(w.salary) AS total_salary
FROM works w
```

```
GROUP BY w.company_name;

SELECT e.ename
FROM emp e
JOIN works w ON e.eid = w.eid
WHERE w.company_name = 'Accenture';
```

EXTRAS

```
SELECT * FROM works;

SELECT * FROM emp;

SELECT * FROM company;
```

ASSIGNMENT 5

Same as ASSIGNMENT 2

ASSIGNMENT 6

Same as ASSIGNMENT 3

ASSIGNMENT 7

Car Rental Database Management System
Customers (CustomerID, Name, Email, Phone, City)
Cars (CarID, Model, Brand, Year, RentalPricePerDay, AvailabilityStatus)
Rentals (RentalID, CustomerID, CarID, StartDate, EndDate, TotalAmount)
Write queries for the following questions:
1. Create a Payments table with attributes: PaymentID, RentalID (FK), PaymentDate, AmountPaid, and PaymentMethod.
2. Update AvailabilityStatus of a car to 'Rented' for a specific CustomerID and CarID. 3. Retrieve Customer Name, Car Model, and Rental StartDate for rentals where RentalPricePerDay is above 1000.
4. Calculate the total rental amount collected per Car Brand.
5. Find the top 3 customers who have spent the most on rentals.

ANSWERS

```
DROP DATABASE IF EXISTS CarRentalDB;

CREATE DATABASE CarRentalDB;
USE CarRentalDB;

CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY AUTO_INCREMENT,
   Name VARCHAR(100),
   Email VARCHAR(100),
   Phone VARCHAR(15),
   City VARCHAR(50)
);

CREATE TABLE Cars (
   CarID INT PRIMARY KEY AUTO_INCREMENT,
```

```
    Model VARCHAR(100),
    Brand VARCHAR(50),
    Year INT,
    RentalPricePerDay DECIMAL(10, 2),
    AvailabilityStatus VARCHAR(20)
);

CREATE TABLE Rentals (
    RentalID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    CarID INT,
    StartDate DATE,
    EndDate DATE,
    TotalAmount DECIMAL(10, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (CarID) REFERENCES Cars(CarID)
);


CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY AUTO_INCREMENT,
    RentalID INT,
    PaymentDate DATE,
    AmountPaid DECIMAL(10, 2),
    PaymentMethod VARCHAR(50),
    FOREIGN KEY (RentalID) REFERENCES Rentals(RentalID)
);

INSERT INTO Customers (Name, Email, Phone, City)
VALUES
('Satyam', 'satyam@gmail.com', '6969696969', 'Pune'),
('Shashank', 'shashank@gmail.com', '9696969696', 'Delhi');

INSERT INTO Cars (Model, Brand, Year, RentalPricePerDay, AvailabilityStatus)
VALUES
('i20', 'Hyundai', 2022, 1200, 'Available'),
('City', 'Honda', 2021, 900, 'Available');

INSERT INTO Rentals (CustomerID, CarID, StartDate, EndDate, TotalAmount)
VALUES
(1, 1, '2025-04-24', '2025-04-25', 2400),
(2, 2, '2025-04-24', '2025-04-25', 1800);

UPDATE Cars
SET AvailabilityStatus = 'Rented'
WHERE CarID = (
    SELECT CarID FROM Rentals
    WHERE CustomerID = 1 AND CarID = 1
);

INSERT INTO Payments (RentalID, PaymentDate, AmountPaid, PaymentMethod)
VALUES
(1, '2025-04-25', 2400, 'UPI'),
(2, '2025-04-25', 1800, 'Credit Card');
```

EXTRAS not asked

SELECT * FROM Customers;

SELECT * FROM Cars;

SELECT * FROM Rentals;

SELECT * FROM Payments;

ASSIGNMENT 8

Online Shopping System
1. Customers (CustomerID, Name, Email, Phone, Address)
2. Products (ProductID, Name, Category, Price, StockQuantity)
3. Orders (OrderID, CustomerID, OrderDate, TotalAmount)
4. OrderDetails (OrderDetailID, OrderID, ProductID, Quantity, Subtotal)
Write queries for the following questions:
1. Create a Payments table with PaymentID, OrderID (FK), PaymentDate, AmountPaid, and PaymentMethod.
2. Update the stock quantity of a product after an order is placed.
3. Retrieve Customer Name, Order Date, and TotalAmount for orders where the total amount exceeds 5000.
4. Calculate the total sales per product category.
5. Find the top 5 customers who have spent the most on orders.

ANSWERS

```
DROP DATABASE IF EXISTS OnlineShoppingDB;
CREATE DATABASE OnlineShoppingDB;
USE OnlineShoppingDB;

CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY AUTO_INCREMENT,
   Name VARCHAR(100),
   Email VARCHAR(100),
   Phone VARCHAR(15),
   Address VARCHAR(200)
);

CREATE TABLE Products (
   ProductID INT PRIMARY KEY AUTO_INCREMENT,
   Name VARCHAR(100),
   Category VARCHAR(50),
   Price DECIMAL(10, 2),
   StockQuantity INT
);

CREATE TABLE Orders (
   OrderID INT PRIMARY KEY AUTO_INCREMENT,
   CustomerID INT,
   OrderDate DATE,
   TotalAmount DECIMAL(10, 2),
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

```
CREATE TABLE OrderDetails (
   OrderDetailID INT PRIMARY KEY AUTO_INCREMENT,
   OrderID INT,
   ProductID INT,
   Quantity INT,
   Subtotal DECIMAL(10, 2),
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

CREATE TABLE Payments (
   PaymentID INT PRIMARY KEY AUTO_INCREMENT,
   OrderID INT,
   PaymentDate DATE,
   AmountPaid DECIMAL(10, 2),
   PaymentMethod VARCHAR(50),
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);

INSERT INTO Customers (Name, Email, Phone, Address)
VALUES
('Satyam', 'satyam@gmail.com', '6969696969', 'Pune, Maharashtra'),
('Shashank', 'shashank@gmail.com', '9696969696', 'Delhi, India');

INSERT INTO Products (Name, Category, Price, StockQuantity)
VALUES
('Laptop', 'Electronics', 50000, 10),
('Mobile Phone', 'Electronics', 30000, 20),
('Shirt', 'Clothing', 1000, 50),
('Washing Machine', 'Home Appliances', 15000, 5);

INSERT INTO Orders (CustomerID, OrderDate, TotalAmount)
VALUES
(1, '2025-04-24', 55000),
(2, '2025-04-23', 12000);

INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Subtotal)
VALUES
(1, 1, 1, 50000),
(1, 2, 1, 30000),
(2, 3, 2, 2000);

INSERT INTO Payments (OrderID, PaymentDate, AmountPaid, PaymentMethod)
VALUES
(1, '2025-04-24', 55000, 'Credit Card'),
(2, '2025-04-23', 12000, 'Debit Card');

-- 2. Query to update the stock quantity of a product after an order is placed
UPDATE Products
SET StockQuantity = StockQuantity - (SELECT Quantity FROM OrderDetails WHERE ProductID = Products.ProductID
AND OrderID = 1)
WHERE ProductID = 1;


-- 3. Query to retrieve Customer Name, Order Date, and TotalAmount for orders where the total amount exceeds
5000
```

```
SELECT cu.Name AS CustomerName, o.OrderDate, o.TotalAmount
FROM Orders o
JOIN Customers cu ON o.CustomerID = cu.CustomerID
WHERE o.TotalAmount > 5000;


-- 4. Query to calculate the total sales per product category
SELECT p.Category, SUM(od.Subtotal) AS TotalSales
FROM OrderDetails od
JOIN Products p ON od.ProductID = p.ProductID
GROUP BY p.Category;

-- 5. Query to find the top 5 customers who have spent the most on orders
SELECT cu.CustomerID, cu.Name, SUM(o.TotalAmount) AS TotalSpent
FROM Customers cu
JOIN Orders o ON cu.CustomerID = o.CustomerID
GROUP BY cu.CustomerID, cu.Name
ORDER BY TotalSpent DESC
LIMIT 5;

SELECT * FROM Customers;
SELECT * FROM Products;
SELECT * FROM Orders;
SELECT * FROM OrderDetails;
SELECT * FROM Payments;
```

ASSIGNMENT 9

Library Management System
1. Members (MemberID, Name, Email, Phone, MembershipDate)
2. Books (BookID, Title, Author, Genre, CopiesAvailable)
3. BorrowedBooks (BorrowID, MemberID, BookID, BorrowDate, ReturnDate) Write queries for the following questions:
1. CREATE TABLE – Create a Fines table with FineID, MemberID (FK), Amount, Status, and FineDate.
2. UPDATE – Update CopiesAvailable when a book is borrowed or returned. 3. SELECT with JOIN & Operators – Retrieve Member Name, Book Title, and Borrow Date for books borrowed in the last month.
4. GROUP BY & Aggregate Function – Find the number of books borrowed per genre. 5. Joins & Aggregate Functions – Find the top 5 members who borrowed the most books.

ANSWERS

```
-- Drop and recreate database
DROP DATABASE IF EXISTS LibraryDB;
CREATE DATABASE LibraryDB;
USE LibraryDB;

-- Create Members table
CREATE TABLE Members (
   MemberID INT PRIMARY KEY AUTO_INCREMENT,
   Name VARCHAR(100),
   Email VARCHAR(100),
   Phone VARCHAR(15),
   MembershipDate DATE
);
```

```sql
-- Create Books table
CREATE TABLE Books (
    BookID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(200),
    Author VARCHAR(100),
    Genre VARCHAR(50),
    CopiesAvailable INT
);

-- Create BorrowedBooks table
CREATE TABLE BorrowedBooks (
    BorrowID INT PRIMARY KEY AUTO_INCREMENT,
    MemberID INT,
    BookID INT,
    BorrowDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID)
);

--  Create Fines table
CREATE TABLE Fines (
    FineID INT PRIMARY KEY AUTO_INCREMENT,
    MemberID INT,
    Amount DECIMAL(10, 2),
    Status VARCHAR(20),
    FineDate DATE,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);

-- Insert Members
INSERT INTO Members (Name, Email, Phone, MembershipDate) VALUES
('Satyam', 'satyam@gmail.com', '1234567890', '2025-01-15'),
('Shashank', 'shashank@gmail.com', '9876543210', '2025-02-20'),
('Saurav', 'saurav@gmail.com', '9988776655', '2025-03-10'),
('Anjali', 'anjali@gmail.com', '7894561230', '2025-04-05');

-- Insert Books
INSERT INTO Books (Title, Author, Genre, CopiesAvailable) VALUES
('The Martian', 'Andy Weir', 'Science Fiction', 3),
('The 3 Mistakes of My Life', 'Chetan Bhagat', 'Fiction', 4),
('My Experiments with Truth', 'Mahatma Gandhi', 'Autobiography', 2);

-- Insert BorrowedBooks
INSERT INTO BorrowedBooks (MemberID, BookID, BorrowDate, ReturnDate) VALUES
(2, 2, '2025-04-25', '2025-04-29'),
(3, 3, '2025-04-26', '2025-04-30'),
(1, 1, '2025-04-27', '2025-04-30'),
(2, 3, '2025-04-28', '2025-04-30'),
(4, 2, '2025-04-29', '2025-04-30');

-- Insert Fines
INSERT INTO Fines (MemberID, Amount, Status, FineDate) VALUES
(1, 20.00, 'Unpaid', '2025-04-30'),
(2, 10.00, 'Paid', '2025-04-30');
```

```
-- When a book is borrowed (e.g., BookID 1)
UPDATE Books
SET CopiesAvailable = CopiesAvailable - 1
WHERE BookID = 1;

-- When a book is returned (e.g., BookID 1)
UPDATE Books
SET CopiesAvailable = CopiesAvailable + 1
WHERE BookID = 1;


-- Retrieve Member Name, Book Title, and Borrow Date for Books Borrowed in the Last Month
SELECT
   m.Name AS MemberName,
   b.Title AS BookTitle,
   bb.BorrowDate
FROM BorrowedBooks bb
JOIN Members m ON bb.MemberID = m.MemberID
JOIN Books b ON bb.BookID = b.BookID
WHERE bb.BorrowDate BETWEEN '2025-04-01' AND '2025-04-30';

-- Find the Number of Books Borrowed Per Genre

SELECT
   b.Genre,
   COUNT(*) AS TotalBorrowed
FROM BorrowedBooks bb
JOIN Books b ON bb.BookID = b.BookID
GROUP BY b.Genre;

-- Find the Top 5 Members Who Borrowed the Most Books

SELECT
   m.Name AS MemberName,
   COUNT(*) AS BooksBorrowed
FROM BorrowedBooks bb
JOIN Members m ON bb.MemberID = m.MemberID
GROUP BY m.MemberID
ORDER BY BooksBorrowed DESC
LIMIT 5;

EXTRAS not asked

SELECT * FROM Members ORDER BY MemberID;

SELECT * FROM Books ORDER BY BookID;

SELECT * FROM BorrowedBooks ORDER BY BorrowDate, MemberID;
```

ASSIGNMENT 10

Hospital Management System
1. Patients (PatientID, Name, Age, Gender, Contact)
2. Doctors (DoctorID, Name, Specialization, Contact)
3. Appointments (AppointmentID, PatientID, DoctorID, AppointmentDate, Status) 4. Bills (BillID, PatientID, Amount, PaymentStatus)

Write queries for the following questions:
1. Create table – create a medicalrecords table with recordid, patientid (fk), diagnosis, prescription, and recorddate.
2. Update – update an appointment status to "completed" after a patient's visit. 3. Select with join & operators – retrieve patient name, doctor name, and appointment date for patients who consulted a specific specialization.
4. Group by & aggregate function – find the total revenue collected per doctor. 5. Joins & aggregate functions – find the top 3 doctors who attended the highest number of appointments.

ANSWERS

```
-- Drop the existing database if it exists
DROP DATABASE IF EXISTS HospitalManagementSystem;

-- Create a new database
CREATE DATABASE HospitalManagementSystem;

-- Use the newly created database
USE HospitalManagementSystem;

-- Create Patients table
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Contact VARCHAR(15)
);

-- Create Doctors table
CREATE TABLE Doctors (
    DoctorID INT PRIMARY KEY,
    Name VARCHAR(100),
    Specialization VARCHAR(100),
    Contact VARCHAR(15)
);

-- Create Appointments table
CREATE TABLE Appointments (
    AppointmentID INT PRIMARY KEY,
    PatientID INT,
    DoctorID INT,
    AppointmentDate DATE,
    Status VARCHAR(50),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);

-- Create Bills table
CREATE TABLE Bills (
    BillID INT PRIMARY KEY,
    PatientID INT,
    Amount DECIMAL(10, 2),
    PaymentStatus VARCHAR(20),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);
```

```
-- Create medicalrecords table
CREATE TABLE medicalrecords (
   recordid INT PRIMARY KEY,
   patientid INT,
   diagnosis TEXT,
   prescription TEXT,
   recorddate DATE,
   FOREIGN KEY (patientid) REFERENCES Patients(PatientID)
);

-- Insert Patients data (with two more patients)
INSERT INTO Patients (PatientID, Name, Age, Gender, Contact)
VALUES
   (1, 'Rahul Sharma', 30, 'Male', '9876543210'),
   (2, 'Rohan Verma', 28, 'Male', '9123456789'),
   (3, 'Priya Deshmukh', 25, 'Female', '9988776655'),
   (4, 'Aisha Khan', 35, 'Female', '9456123789');

-- Insert Doctors data (with two more doctors)
INSERT INTO Doctors (DoctorID, Name, Specialization, Contact)
VALUES
   (1, 'Dr. Panda', 'Cardiology', '9988776655'),
   (2, 'Dr. Rajesh Kumar', 'Dermatology', '8877665544'),
   (3, 'Dr. Vivek Sharma', 'Neurology', '9966332211'),
   (4, 'Dr. Anjali Mehta', 'Orthopedics', '9977445566'),
   (5, 'Dr. Sanjay Gupta', 'Physician', '9966887744');

-- Insert Appointments data
INSERT INTO Appointments (AppointmentID, PatientID, DoctorID, AppointmentDate, Status)
VALUES
   (1, 1, 1, '2025-04-23', 'scheduled'),
   (2, 2, 2, '2025-04-24', 'scheduled'),
   (3, 3, 4, '2025-04-25', 'scheduled'),
   (4, 4, 5, '2025-04-26', 'scheduled');

-- Insert Bills data
INSERT INTO Bills (BillID, PatientID, Amount, PaymentStatus)
VALUES
   (1, 1, 150.00, 'Paid'),
   (2, 2, 200.00, 'Unpaid'),
   (3, 3, 100.00, 'Paid'),
   (4, 4, 250.00, 'Unpaid');

-- Insert medicalrecords data with corrected unique recordids
INSERT INTO medicalrecords (recordid, patientid, diagnosis, prescription, recorddate)
VALUES
   (3, 1, 'Hypertension', 'Paracetamol 5mg', '2025-04-23'),
   (4, 2, 'Acne', 'Benzoyl Peroxide 5%', '2025-04-24'),
   (5, 3, 'Sprained Ankle', 'Ibuprofen 400mg', '2025-04-25'),
   (6, 4, 'Sore Throat', 'Dolo 650', '2025-04-26');

-- Update an appointment status to "completed" after a patient's visit
UPDATE Appointments
SET Status = 'completed'
WHERE AppointmentID = 1;
```

```
-- Retrieve patient name, doctor name, and appointment date for patients who consulted a specific specialization
SELECT
Patients.Name AS PatientName,
Doctors.Name AS DoctorName,
Appointments.AppointmentDate
FROM Appointments
JOIN Patients ON Appointments.PatientID = Patients.PatientID
JOIN Doctors ON Appointments.DoctorID = Doctors.DoctorID
WHERE Doctors.Specialization = 'Cardiology';

-- Find the total revenue collected per doctor
SELECT
Doctors.Name AS DoctorName,
SUM(Bills.Amount) AS TotalRevenue
FROM Bills
JOIN Patients ON Bills.PatientID = Patients.PatientID
JOIN Appointments ON Patients.PatientID = Appointments.PatientID
JOIN Doctors ON Appointments.DoctorID = Doctors.DoctorID
GROUP BY Doctors.Name;

-- Find the top 3 doctors who attended the highest number of appointments
SELECT
Doctors.Name AS DoctorName,
COUNT(Appointments.AppointmentID) AS AppointmentCount
FROM Appointments
JOIN Doctors ON Appointments.DoctorID = Doctors.DoctorID
GROUP BY Doctors.Name
ORDER BY AppointmentCount DESC
LIMIT 3;
```

ASSIGNMENT 11

University Database Management System
1. Student Management: Store student details such as StudentID, Name, Age, Gender, Department, and Email.
2. Course Management: Maintain course details including CourseID, CourseName, Credits, and Department.
3. Enrollment System: Allow students to enroll in multiple courses, tracking StudentID, CourseID, EnrollmentDate, and Grade.
4. Professor Management: Store professor details like ProfessorID, Name, Department, and Email
Write queries for the following questions:
1. Calculate percentage of students in each department
2. Detect duplicate enrollments (same student enrolled in same course in the same semester) 3. Find the semester with the highest average enrollments per course
4. List students with more than 3 enrollments
5. List all courses and the number of students enrolled in each

ANSWERS

```
-- Drop the existing UniversityDB database if it exists
DROP DATABASE IF EXISTS UniversityDB;

-- Create a new database named UniversityDB
CREATE DATABASE UniversityDB;
```

```sql
-- Switch to using UniversityDB
USE UniversityDB;

-- Create Students table with relevant columns
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   Age INT,
   Gender VARCHAR(10),
   Department VARCHAR(100),
   Email VARCHAR(100)
);

-- Create Courses table to store course info
CREATE TABLE Courses (
   CourseID INT PRIMARY KEY,
   CourseName VARCHAR(100),
   Credits INT,
   Department VARCHAR(100)
);

-- Create Enrollments table to track which student enrolled in which course
CREATE TABLE Enrollments (
   EnrollmentID INT PRIMARY KEY,
   StudentID INT,
   CourseID INT,
   EnrollmentDate DATE,
   Grade VARCHAR(5),
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

-- Create Professors table to manage professor data
CREATE TABLE Professors (
   ProfessorID INT PRIMARY KEY,
   Name VARCHAR(100),
   Department VARCHAR(100),
   Email VARCHAR(100)
);

-- Insert sample students
INSERT INTO Students (StudentID, Name, Age, Gender, Department, Email)
VALUES
(1, 'Satyam Bhatt', 21, 'Male', 'Computer Science', 'satyam@gmail.com'),
(2, 'Shashank Raj', 22, 'Male', 'Electrical Engineering', 'shashank@gmail.com'),
(3, 'Saurav Kumar', 20, 'Male', 'Mechanical Engineering', 'saurav@gmail.com'),
(4, 'Adarsh Kumar', 23, 'Male', 'Computer Science', 'adarsh@gmail.com'),
(5, 'Priya Singh', 21, 'Female', 'Mathematics', 'priya@gmail.com');

-- Insert sample courses
INSERT INTO Courses (CourseID, CourseName, Credits, Department)
VALUES
(101, 'Data Structures', 4, 'Computer Science'),
(102, 'Electrical Machines', 3, 'Electrical Engineering'),
(103, 'Thermodynamics', 3, 'Mechanical Engineering'),
(104, 'Linear Algebra', 4, 'Mathematics');
```

```
-- Insert sample professors
INSERT INTO Professors (ProfessorID, Name, Department, Email)
VALUES
(1, 'Dr. Anjali Mehta', 'Computer Science', 'anjali@university.in'),
(2, 'Dr. Ravi Shankar', 'Electrical Engineering', 'ravi@university.in'),
(3, 'Dr. Neha Kapoor', 'Mathematics', 'neha@university.in');

-- Insert enrollment data for students
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID, EnrollmentDate, Grade)
VALUES
(1, 1, 101, '2025-01-10', 'A'),
(2, 2, 102, '2025-01-11', 'B'),
(3, 3, 103, '2025-01-12', 'C'),
(4, 4, 101, '2025-01-13', 'B'),
(5, 5, 104, '2025-01-14', 'A'),
(6, 1, 102, '2025-01-15', 'B'),
(7, 1, 103, '2025-01-16', 'A'),
(8, 1, 104, '2025-01-17', 'A'),
(9, 2, 101, '2025-01-18', 'C'),
(10, 1, 101, '2025-01-19', 'A'); -- Duplicate enrollment for student 1 in course 101

-- Query 1: Percentage of students in each department
SELECT Department,
(COUNT() * 100.0 / (SELECT COUNT() FROM Students)) AS Percentage
FROM Students
GROUP BY Department;

-- Query 2: Detect duplicate enrollments (same student in same course more than once)
SELECT StudentID, CourseID, COUNT(*) AS TimesEnrolled
FROM Enrollments
GROUP BY StudentID, CourseID
HAVING COUNT(*) > 1;

-- Query 3: Find semester (year/month) with highest average enrollments per course
SELECT YEAR(EnrollmentDate) AS Year, MONTH(EnrollmentDate) AS Month,
COUNT(*) / COUNT(DISTINCT CourseID) AS AvgEnrollmentsPerCourse
FROM Enrollments
GROUP BY YEAR(EnrollmentDate), MONTH(EnrollmentDate)
ORDER BY AvgEnrollmentsPerCourse DESC
LIMIT 1;

-- Query 4: List students with more than 3 enrollments
SELECT s.StudentID, s.Name, COUNT(e.EnrollmentID) AS TotalEnrollments
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
GROUP BY s.StudentID, s.Name
HAVING COUNT(e.EnrollmentID) > 3;

-- Query 5: List all courses and number of students enrolled in each
SELECT c.CourseName, COUNT(e.StudentID) AS NumberOfStudents
FROM Courses c
LEFT JOIN Enrollments e ON c.CourseID = e.CourseID
GROUP BY c.CourseName;

-- View all student records
SELECT * FROM Students;
```

```
-- View all course records
SELECT * FROM Courses;

-- View all professor records
SELECT * FROM Professors;

-- View all enrollment records
SELECT * FROM Enrollments;
```

ASSIGNMENT 12

University Database Management System
1. Student Management: Store student details such as StudentID, Name, Age, Gender, Department, and Email.
2. Course Management: Maintain course details including CourseID, CourseName, Credits, and Department.
3. Enrollment System: Allow students to enroll in multiple courses, tracking StudentID, CourseID, EnrollmentDate, and Grade.
4. Professor Management: Store professor details like ProfessorID, Name, Department, and Email
Write queries for the following questions:
1. List all courses a specific student is enrolled in (e.g., Pooja)
2. Identify students who failed more than 2 courses (assuming grade < 2.0 is fail) 3. Count the number of students in each department
4. Find courses with zero enrollments
5. Find the most popular course (course with the highest number of enrollments)

ANSWERS

```
-- 1. Drop the existing database if it exists
DROP DATABASE IF EXISTS UniversityDB;

-- 2. Create a new database
CREATE DATABASE UniversityDB;

-- 3. Select the database for use
USE UniversityDB;

-- 4. Create the Students table
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(100),
    Email VARCHAR(100)
);

-- 5. Create the Courses table
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
```

```
    Credits INT,
    Department VARCHAR(100)
);

-- 6. Create the Enrollments table
CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    Grade VARCHAR(5),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

-- 7. Create the Professors table
CREATE TABLE Professors (
    ProfessorID INT PRIMARY KEY,
    Name VARCHAR(100),
    Department VARCHAR(100),
    Email VARCHAR(100)
);

-- 8. Insert sample data into Students
INSERT INTO Students (StudentID, Name, Age, Gender, Department, Email)
VALUES
(1, 'Satyam Bhatt', 21, 'Male', 'Computer Science', 'satyam@gmail.com'),
(2, 'Shashank Raj', 22, 'Male', 'Electrical Engineering', 'shashank@gmail.com'),
(3, 'Saurav Kumar', 20, 'Male', 'Mechanical Engineering', 'saurav@gmail.com'),
(4, 'Adarsh Kumar', 23, 'Male', 'Computer Science', 'adarsh@gmail.com'),
(5, 'Priya Singh', 21, 'Female', 'Mathematics', 'priya@gmail.com'),
(6, 'Pooja Verma', 22, 'Female', 'Computer Science', 'pooja@gmail.com'); -- For later query

-- 9. Insert sample data into Courses
INSERT INTO Courses (CourseID, CourseName, Credits, Department)
VALUES
(101, 'Data Structures', 4, 'Computer Science'),
(102, 'Electrical Machines', 3, 'Electrical Engineering'),
(103, 'Thermodynamics', 3, 'Mechanical Engineering'),
(104, 'Linear Algebra', 4, 'Mathematics'),
(105, 'Digital Logic', 3, 'Computer Science'); -- No enrollment for test

-- 10. Insert sample data into Professors
INSERT INTO Professors (ProfessorID, Name, Department, Email)
VALUES
(1, 'Dr. Anjali Mehta', 'Computer Science', 'anjali@university.in'),
(2, 'Dr. Ravi Shankar', 'Electrical Engineering', 'ravi@university.in'),
(3, 'Dr. Neha Kapoor', 'Mathematics', 'neha@university.in');

-- 11. Insert sample data into Enrollments
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID, EnrollmentDate, Grade)
VALUES
(1, 1, 101, '2025-01-10', 'A'),
(2, 2, 102, '2025-01-11', 'B'),
(3, 3, 103, '2025-01-12', 'C'),
(4, 4, 101, '2025-01-13', 'D'), -- Grade D (fail)
(5, 5, 104, '2025-01-14', 'A'),
```

```
(6, 1, 102, '2025-01-15', 'B'),
(7, 1, 103, '2025-01-16', 'F'), -- Grade F (fail)
(8, 6, 101, '2025-01-17', 'F'), -- Grade F (fail)
(9, 6, 102, '2025-01-18', 'F'), -- Grade F (fail)
(10, 6, 103, '2025-01-19', 'D'); -- Grade D (fail) — total 4 fails for Pooja



-- Q1. List all courses a specific student is enrolled in (e.g., Pooja)
SELECT s.Name AS StudentName, c.CourseName
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
JOIN Courses c ON e.CourseID = c.CourseID
WHERE s.Name = 'Pooja Verma';

-- Q2. Identify students who failed more than 2 courses (assuming grade < 2.0 is fail)
SELECT s.StudentID, s.Name, COUNT(*) AS FailedCourses
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
WHERE CASE e.Grade
WHEN 'A' THEN 4.0
WHEN 'B' THEN 3.0
WHEN 'C' THEN 2.0
WHEN 'D' THEN 1.0
WHEN 'F' THEN 0.0
ELSE NULL
END < 2.0
GROUP BY s.StudentID, s.Name
HAVING COUNT(*) > 2;

-- Q3. Count the number of students in each department
SELECT Department, COUNT(*) AS NumberOfStudents
FROM Students
GROUP BY Department;

-- Q4. Find courses with zero enrollments
SELECT c.CourseID, c.CourseName
FROM Courses c
LEFT JOIN Enrollments e ON c.CourseID = e.CourseID
WHERE e.EnrollmentID IS NULL;

-- Q5. Find the most popular course (course with the highest number of enrollments)
SELECT c.CourseID, c.CourseName, COUNT(e.StudentID) AS EnrollmentCount
FROM Courses c
JOIN Enrollments e ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.CourseName
ORDER BY EnrollmentCount DESC
LIMIT 1;

EXTRAS not asked

SELECT * FROM Students;

SELECT * FROM Courses;

SELECT * FROM Professors;
```

```
SELECT * FROM Enrollments;
```

ASSIGNMENT 13

University Database Management System
1. Student Management: Store student details such as StudentID, Name, Age, Gender, Department, and Email.
2. Course Management: Maintain course details including CourseID, CourseName, Credits, and Department.
3. Enrollment System: Allow students to enroll in multiple courses, tracking StudentID, CourseID, EnrollmentDate, and Grade.
4. Professor Management: Store professor details like ProfessorID, Name, Department, and Email
Write queries for the following questions:
1. Find students who have not enrolled in any course
2. Find students who are enrolled in more than 3 courses
3. Find the average grade of students per course
4. Retrieve the highest grade in each course
5. Get the department with the highest number of students

```
-- Drop the existing database if it exists
DROP DATABASE IF EXISTS UniversityDB;

-- Create a new database
CREATE DATABASE UniversityDB;

-- Use the newly created database
USE UniversityDB;

-- Create Students table
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(100),
    Email VARCHAR(100)
);

-- Create Courses table
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Credits INT,
    Department VARCHAR(100)
);

-- Create Enrollments table
CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
```

```
    CourseID INT,
    EnrollmentDate DATE,
    Grade VARCHAR(5),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

-- Create Professors table
CREATE TABLE Professors (
    ProfessorID INT PRIMARY KEY,
    Name VARCHAR(100),
    Department VARCHAR(100),
    Email VARCHAR(100)
);

-- Insert sample data into Students table
INSERT INTO Students (StudentID, Name, Age, Gender, Department, Email)
VALUES
(1, 'Satyam Bhatt', 21, 'Male', 'Computer Science', 'satyam@gmail.com'),
(2, 'Shashank Raj', 22, 'Male', 'Electrical Engineering', 'shashank@gmail.com'),
(3, 'Saurav Kumar', 20, 'Male', 'Mechanical Engineering', 'saurav@gmail.com'),
(4, 'Adarsh Kumar', 23, 'Male', 'Computer Science', 'adarsh@gmail.com'),
(5, 'Priya Singh', 21, 'Female', 'Mathematics', 'priya@gmail.com');

-- Insert sample data into Courses table
INSERT INTO Courses (CourseID, CourseName, Credits, Department)
VALUES
(101, 'Data Structures', 4, 'Computer Science'),
(102, 'Electrical Machines', 3, 'Electrical Engineering'),
(103, 'Thermodynamics', 3, 'Mechanical Engineering'),
(104, 'Linear Algebra', 4, 'Mathematics');

-- Insert sample data into Professors table
INSERT INTO Professors (ProfessorID, Name, Department, Email)
VALUES
(1, 'Dr. Anjali Mehta', 'Computer Science', 'anjali@university.in'),
(2, 'Dr. Ravi Shankar', 'Electrical Engineering', 'ravi@university.in'),
(3, 'Dr. Neha Kapoor', 'Mathematics', 'neha@university.in');

-- Insert sample data into Enrollments table
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID, EnrollmentDate, Grade)
VALUES
(1, 1, 101, '2025-01-10', 'A'),
(2, 2, 102, '2025-01-11', 'B'),
(3, 3, 103, '2025-01-12', 'C'),
(4, 4, 101, '2025-01-13', 'B'),
(5, 5, 104, '2025-01-14', 'A'),
(6, 1, 102, '2025-01-15', 'B'),
(7, 1, 103, '2025-01-16', 'A'),
(8, 1, 104, '2025-01-17', 'A'),
(9, 2, 101, '2025-01-18', 'C'),
(10, 1, 101, '2025-01-19', 'A');

-- Query to find students who have not enrolled in any course
SELECT s.StudentID, s.Name
FROM Students s
LEFT JOIN Enrollments e ON s.StudentID = e.StudentID
```

```
WHERE e.CourseID IS NULL;

-- Query to find students who are enrolled in more than 3 courses
SELECT s.StudentID, s.Name, COUNT(e.CourseID) AS CourseCount
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
GROUP BY s.StudentID, s.Name
HAVING COUNT(e.CourseID) > 3;

-- Query to find the average grade of students per course
SELECT c.CourseName,
    AVG(
      CASE Grade
        WHEN 'A' THEN 4
        WHEN 'B' THEN 3
        WHEN 'C' THEN 2
        WHEN 'D' THEN 1
        WHEN 'F' THEN 0
      END
    ) AS AverageGrade
FROM Enrollments e
JOIN Courses c ON e.CourseID = c.CourseID
GROUP BY c.CourseName;

-- Query to retrieve the highest grade in each course
SELECT c.CourseName,
    MAX(
      CASE Grade
        WHEN 'A' THEN 4
        WHEN 'B' THEN 3
        WHEN 'C' THEN 2
        WHEN 'D' THEN 1
        WHEN 'F' THEN 0
      END
    ) AS MaxGradeNumeric
FROM Enrollments e
JOIN Courses c ON e.CourseID = c.CourseID
GROUP BY c.CourseName;

-- Query to get the department with the highest number of students
SELECT Department, COUNT(*) AS StudentCount
FROM Students
GROUP BY Department
ORDER BY StudentCount DESC
LIMIT 1;


EXTRAS not asked

-- Display all student data
SELECT * FROM Students;

-- Display all course data
SELECT * FROM Courses;

-- Display all enrollment data
SELECT * FROM Enrollments;
```

```
-- Display all professor data
SELECT * FROM Professors;
```

ASSIGNMENT 14

Bank database Management System

1. Customer (customer_id, name, address, phone, email)
2. Account (account_id, customer_id, account_type, balance, branch_id)
3. Branch (branch_id, branch_name, location, manager_id)
4. Transaction (transaction_id, account_id, transaction_type, amount, transaction_date) 5. Loan (loan_id, customer_id, amount, loan_type, status)
6. Employee (employee_id, name, position, branch_id, salary)

Write queries for the following questions:

1. List all customers and their account details
2. Find the total balance in each branch
3. Find customers who have taken loans greater than Rs. 1,00,000
4. Retrieve transaction history for a specific account (e.g., Account ID: 101)
5. Find customers who have both a loan and an account
6. Create a view of high-value customers (balance > 1,00,000)

ANSWERS

```
-- Drop the existing database if it exists
DROP DATABASE IF EXISTS BankDB;

-- Create a new database
CREATE DATABASE BankDB;

-- Use the newly created database
USE BankDB;

-- Create Customer table
CREATE TABLE Customer (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    address VARCHAR(200),
    phone VARCHAR(15),
    email VARCHAR(100)
);

-- Create Account table
CREATE TABLE Account (
    account_id INT PRIMARY KEY,
    customer_id INT,
    account_type VARCHAR(50),
    balance DECIMAL(15, 2),
    branch_id INT,
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

-- Create Branch table
CREATE TABLE Branch (
```

```
    branch_id INT PRIMARY KEY,
    branch_name VARCHAR(100),
    location VARCHAR(100),
    manager_id INT
);

-- Create Transaction table
CREATE TABLE Transaction (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    transaction_type VARCHAR(50),
    amount DECIMAL(15, 2),
    transaction_date DATE,
    FOREIGN KEY (account_id) REFERENCES Account(account_id)
);

-- Create Loan table
CREATE TABLE Loan (
    loan_id INT PRIMARY KEY,
    customer_id INT,
    amount DECIMAL(15, 2),
    loan_type VARCHAR(50),
    status VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

-- Create Employee table
CREATE TABLE Employee (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    position VARCHAR(50),
    branch_id INT,
    salary DECIMAL(15, 2),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);

-- Insert sample data into Customer table
INSERT INTO Customer (customer_id, name, address, phone, email) VALUES
(1, 'Satyam Bhatt', 'Pune', '9999999991', 'satyam@gmail.com'),
(2, 'Adarsh Kumar', 'Mumbai', '9999999992', 'adarsh@gmail.com'),
(3, 'Priya Singh', 'Delhi', '9999999993', 'priya@gmail.com');

-- Insert sample data into Branch table
INSERT INTO Branch (branch_id, branch_name, location, manager_id) VALUES
(1, 'Main Branch', 'Pune', 101),
(2, 'City Branch', 'Mumbai', 102);

-- Insert sample data into Account table
INSERT INTO Account (account_id, customer_id, account_type, balance, branch_id) VALUES
(101, 1, 'Savings', 150000, 1),
(102, 2, 'Current', 75000, 2),
(103, 3, 'Savings', 120000, 1);

-- Insert sample data into Transaction table
INSERT INTO Transaction (transaction_id, account_id, transaction_type, amount, transaction_date) VALUES
(1, 101, 'Deposit', 50000, '2025-01-01'),
(2, 101, 'Withdrawal', 10000, '2025-01-05'),
```

```
(3, 102, 'Deposit', 20000, '2025-01-10'),
(4, 103, 'Deposit', 120000, '2025-01-12');

-- Insert sample data into Loan table
INSERT INTO Loan (loan_id, customer_id, amount, loan_type, status) VALUES
(1, 1, 200000, 'Home', 'Approved'),
(2, 3, 50000, 'Personal', 'Pending');

-- Insert sample data into Employee table
INSERT INTO Employee (employee_id, name, position, branch_id, salary) VALUES
(101, 'Ravi Mehta', 'Manager', 1, 90000),
(102, 'Neha Sharma', 'Manager', 2, 85000);

-- 1. List all customers and their account details
SELECT c.customer_id, c.name, a.account_id, a.account_type, a.balance, a.branch_id
FROM Customer c
JOIN Account a ON c.customer_id = a.customer_id;

-- 2. Find the total balance in each branch
SELECT b.branch_name, SUM(a.balance) AS total_balance
FROM Branch b
JOIN Account a ON b.branch_id = a.branch_id
GROUP BY b.branch_name;

-- 3. Find customers who have taken loans greater than Rs. 1,00,000
SELECT c.customer_id, c.name, l.amount
FROM Customer c
JOIN Loan l ON c.customer_id = l.customer_id
WHERE l.amount > 100000;

-- 4. Retrieve transaction history for a specific account (e.g., Account ID: 101)
SELECT t.transaction_id, t.transaction_type, t.amount, t.transaction_date
FROM Transaction t
WHERE t.account_id = 101;

-- 5. Find customers who have both a loan and an account
SELECT DISTINCT c.customer_id, c.name
FROM Customer c
JOIN Account a ON c.customer_id = a.customer_id
JOIN Loan l ON c.customer_id = l.customer_id;

-- 6. Create a view of high-value customers (balance > 1,00,000)
CREATE VIEW HighValueCustomers AS
SELECT c.customer_id, c.name, a.account_id, a.balance
FROM Customer c
JOIN Account a ON c.customer_id = a.customer_id
WHERE a.balance > 100000;


EXTRAS not asked

-- Display all customer data
SELECT * FROM Customer;

-- Display all account data
SELECT * FROM Account;
```

```
-- Display all branch data
SELECT * FROM Branch;

-- Display all transaction data
SELECT * FROM Transaction;

-- Display all loan data
SELECT * FROM Loan;

-- Display all employee data
SELECT * FROM Employee;
```

ASSIGNMENT 15

Bank database Management System

1. Customer (customer_id, name, address, phone, email)
2. Account (account_id, customer_id, account_type, balance, branch_id)
3. Branch (branch_id, branch_name, location, manager_id)
4. Transaction (transaction_id, account_id, transaction_type, amount, transaction_date) 5. Loan (loan_id, customer_id, amount, loan_type, status)
6. Employee (employee_id, name, position, branch_id, salary)

Write queries for the following questions:

1. Find employees working in a specific branch (e.g., Branch ID: 3)
2. Get the details of the highest transaction made
3. Find accounts with a balance less than Rs. 5000
4. Update account balance after a deposit of Rs. 2000 in account ID 105
5. Delete inactive loan applications (status = 'Rejected')
6. Calculate the total loan amount per loan type

ANSWERS

```
-- Drop the existing database if it exists
DROP DATABASE IF EXISTS BankDB;

-- Create a new database
CREATE DATABASE BankDB;

-- Use the newly created database
USE BankDB;

-- Create Customer table
CREATE TABLE Customer (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    address VARCHAR(200),
    phone VARCHAR(15),
    email VARCHAR(100)
);

-- Create Account table
CREATE TABLE Account (
    account_id INT PRIMARY KEY,
    customer_id INT,
```

```
    account_type VARCHAR(50),
    balance DECIMAL(15, 2),
    branch_id INT,
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

-- Create Branch table
CREATE TABLE Branch (
    branch_id INT PRIMARY KEY,
    branch_name VARCHAR(100),
    location VARCHAR(100),
    manager_id INT
);

-- Create Transaction table
CREATE TABLE Transaction (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    transaction_type VARCHAR(50),
    amount DECIMAL(15, 2),
    transaction_date DATE,
    FOREIGN KEY (account_id) REFERENCES Account(account_id)
);

-- Create Loan table
CREATE TABLE Loan (
    loan_id INT PRIMARY KEY,
    customer_id INT,
    amount DECIMAL(15, 2),
    loan_type VARCHAR(50),
    status VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

-- Create Employee table
CREATE TABLE Employee (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    position VARCHAR(100),
    branch_id INT,
    salary DECIMAL(15, 2),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);

-- Insert sample data into Customer
INSERT INTO Customer (customer_id, name, address, phone, email)
VALUES
(1, 'Satyam Bhatt', 'Pune', '9999999999', 'satyam@gmail.com'),
(2, 'Adarsh Kumar', 'Delhi', '8888888888', 'adarsh@gmail.com'),
(3, 'Priya Sharma', 'Mumbai', '7777777777', 'priya@gmail.com');

-- Insert sample data into Branch
INSERT INTO Branch (branch_id, branch_name, location, manager_id)
VALUES
(1, 'Main Branch', 'Pune', 101),
(2, 'North Branch', 'Delhi', 102),
(3, 'South Branch', 'Mumbai', 103);
```

```
-- Insert sample data into Employee
INSERT INTO Employee (employee_id, name, position, branch_id, salary)
VALUES
(201, 'Ravi Verma', 'Clerk', 1, 30000),
(202, 'Neha Singh', 'Manager', 2, 50000),
(203, 'Karan Patel', 'Clerk', 3, 32000);

-- Insert sample data into Account
INSERT INTO Account (account_id, customer_id, account_type, balance, branch_id)
VALUES
(101, 1, 'Savings', 15000.00, 1),
(102, 2, 'Current', 4000.00, 2),
(105, 1, 'Savings', 10000.00, 3);

-- Insert sample data into Transaction
INSERT INTO Transaction (transaction_id, account_id, transaction_type, amount, transaction_date)
VALUES
(1, 101, 'Deposit', 5000.00, '2025-01-10'),
(2, 102, 'Withdrawal', 1000.00, '2025-01-11'),
(3, 105, 'Deposit', 12000.00, '2025-01-12');

-- Insert sample data into Loan
INSERT INTO Loan (loan_id, customer_id, amount, loan_type, status)
VALUES
(1, 1, 200000.00, 'Home', 'Approved'),
(2, 2, 50000.00, 'Education', 'Rejected'),
(3, 3, 150000.00, 'Car', 'Approved');

-- 1. Find employees working in a specific branch (e.g., Branch ID: 3)
SELECT * FROM Employee
WHERE branch_id = 3;

-- 2. Get the details of the highest transaction made
SELECT * FROM Transaction
ORDER BY amount DESC
LIMIT 1;

-- 3. Find accounts with a balance less than Rs. 5000
SELECT * FROM Account
WHERE balance < 5000;

-- 4. Update account balance after a deposit of Rs. 2000 in account ID 105
UPDATE Account
SET balance = balance + 2000
WHERE account_id = 105;

-- 5. Delete inactive loan applications (status = 'Rejected')
DELETE FROM Loan
WHERE status = 'Rejected';

-- Update the balance
UPDATE Account
SET balance = balance + 2000
WHERE account_id = 105;

-- Save the changes if not autocommitted
```

```
COMMIT;

-- Check the updated account
SELECT * FROM Account
WHERE account_id = 105;
```

EXTRAS not asked

```
-- Display all customer data
SELECT * FROM Customer;

-- Display all account data
SELECT * FROM Account;

-- Display all branch data
SELECT * FROM Branch;

-- Display all transaction data
SELECT * FROM Transaction;

-- Display all loan data
SELECT * FROM Loan;

-- Display all employee data
SELECT * FROM Employee;
```

After MidTerm PS:

Problem 1: College Admission System
Schema:
- **Student(sid INT, name VARCHAR(50), gender VARCHAR(10), dept_id INT)**
- **Department(dept_id INT, dept_name VARCHAR(50), intake INT)**

Questions:
1. Create tables with appropriate keys and constraints.
2. Add 5 students and 3 departments.
3. Display names of all male students and their department names.
4. List departments with more than 2 students using GROUP BY and HAVING.
5. Update the intake to increase by 10% for all departments.

```
-- 1. Create tables
CREATE TABLE Department (
   dept_id INT PRIMARY KEY,
   dept_name VARCHAR(50),
   intake INT
);

CREATE TABLE Student (
   sid INT PRIMARY KEY,
   name VARCHAR(50),
   gender VARCHAR(10),
   dept_id INT,
   FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
```

```
);

-- 2. Insert data
INSERT INTO Department VALUES
(1, 'Computer Science', 60),
(2, 'Mechanical', 50),
(3, 'Electronics', 40);

INSERT INTO Student VALUES
(101, 'Aman', 'Male', 1),
(102, 'Riya', 'Female', 2),
(103, 'Rahul', 'Male', 1),
(104, 'Sneha', 'Female', 3),
(105, 'Rohan', 'Male', 2);

-- 3. Display male students and their department names
SELECT s.name, d.dept_name
FROM Student s
JOIN Department d ON s.dept_id = d.dept_id
WHERE s.gender = 'Male';

-- 4. Departments with more than 2 students
SELECT d.dept_name, COUNT(*) AS student_count
FROM Student s
JOIN Department d ON s.dept_id = d.dept_id
GROUP BY d.dept_name
HAVING COUNT(*) > 2;

-- 5. Increase intake by 10%
UPDATE Department
SET intake = intake * 1.1;
```

Problem 2: Online Retail Store
Schema:
- **Customers(cust_id INT, name VARCHAR(50), city VARCHAR(30))**
- **Orders(order_id INT, cust_id INT, amount DECIMAL(10,2), order_date DATE)**

Questions:
1. Create both tables with appropriate constraints.
2. Insert at least 4 customers and 5 orders.
3. Display customer names who placed orders above ₹5000.
4. List total order amount placed by each customer in descending order.
5. Retrieve customers who haven't placed any orders.

```
-- 1. Create tables
CREATE TABLE Customers (
   cust_id INT PRIMARY KEY,
   name VARCHAR(50),
   city VARCHAR(30)
);

CREATE TABLE Orders (
   order_id INT PRIMARY KEY,
   cust_id INT,
   amount DECIMAL(10,2),
```

```
    order_date DATE,
    FOREIGN KEY (cust_id) REFERENCES Customers(cust_id)
);

-- 2. Insert data
INSERT INTO Customers VALUES
(1, 'Anjali', 'Mumbai'),
(2, 'Vikram', 'Delhi'),
(3, 'Neha', 'Pune'),
(4, 'Amit', 'Bangalore');

INSERT INTO Orders VALUES
(101, 1, 6500.00, '2024-03-12'),
(102, 2, 4800.00, '2024-03-15'),
(103, 1, 2300.00, '2024-04-01'),
(104, 3, 5500.00, '2024-04-05'),
(105, 2, 7800.00, '2024-04-10');

-- 3. Customers with orders above ₹5000
SELECT DISTINCT c.name
FROM Customers c
JOIN Orders o ON c.cust_id = o.cust_id
WHERE o.amount > 5000;

-- 4. Total order amount per customer
SELECT c.name, SUM(o.amount) AS total_spent
FROM Customers c
JOIN Orders o ON c.cust_id = o.cust_id
GROUP BY c.name
ORDER BY total_spent DESC;

-- 5. Customers with no orders
SELECT name
FROM Customers
WHERE cust_id NOT IN (
    SELECT DISTINCT cust_id FROM Orders
);
```

Problem 3: Bookstore Inventory
Schema:
- **Books(book_id INT, title VARCHAR(100), price DECIMAL(8,2), pub_year INT)**
- **Sales(sale_id INT, book_id INT, quantity INT, sale_date DATE)**

Questions:
1. Create tables with suitable constraints.
2. Insert 4 books and 5 sales records.
3. Display titles of books sold in the year 2024.
4. Show total sales revenue for each book using SUM(price * quantity).
5. Find the title of the most sold book using ORDER BY and LIMIT.

```
-- 1. Create tables
CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    title VARCHAR(100),
    price DECIMAL(8,2),
    pub_year INT
```

```
);

CREATE TABLE Sales (
   sale_id INT PRIMARY KEY,
   book_id INT,
   quantity INT,
   sale_date DATE,
   FOREIGN KEY (book_id) REFERENCES Books(book_id)
);

-- 2. Insert data
INSERT INTO Books VALUES
(1, 'Learn SQL', 500.00, 2022),
(2, 'Mastering Python', 750.00, 2023),
(3, 'Java Basics', 600.00, 2021),
(4, 'Data Structures', 850.00, 2024);

INSERT INTO Sales VALUES
(201, 1, 5, '2024-02-10'),
(202, 2, 3, '2024-03-12'),
(203, 4, 6, '2024-04-05'),
(204, 2, 2, '2024-01-20'),
(205, 3, 4, '2023-12-25');

-- 3. Titles of books sold in 2024
SELECT DISTINCT b.title
FROM Books b
JOIN Sales s ON b.book_id = s.book_id
WHERE YEAR(s.sale_date) = 2024;

-- 4. Total sales revenue per book
SELECT b.title, SUM(b.price * s.quantity) AS revenue
FROM Books b
JOIN Sales s ON b.book_id = s.book_id
GROUP BY b.title;

-- 5. Most sold book
SELECT b.title
FROM Books b
JOIN Sales s ON b.book_id = s.book_id
GROUP BY b.title
ORDER BY SUM(s.quantity) DESC
LIMIT 1;
```

Problem 4: Airline Reservation
Schema:
- **Flights(flight_id INT, source VARCHAR(30), destination VARCHAR(30), fare DECIMAL(6,2))**
- **Passengers(pid INT, name VARCHAR(50), flight_id INT, travel_date DATE)**

Questions:
1. Create both tables with constraints.
2. Insert 3 flights and 5 passenger bookings.
3. List all passengers travelling to 'Delhi'.
4. Show flight-wise passenger count.

5. Increase fare by 10% for flights having more than 2 bookings.

```
-- 1. Create tables
CREATE TABLE Flights (
   flight_id INT PRIMARY KEY,
   source VARCHAR(30),
   destination VARCHAR(30),
   fare DECIMAL(6,2)
);

CREATE TABLE Passengers (
   pid INT PRIMARY KEY,
   name VARCHAR(50),
   flight_id INT,
   travel_date DATE,
   FOREIGN KEY (flight_id) REFERENCES Flights(flight_id)
);

-- 2. Insert data
INSERT INTO Flights VALUES
(1, 'Mumbai', 'Delhi', 3500.00),
(2, 'Chennai', 'Kolkata', 4200.00),
(3, 'Bangalore', 'Delhi', 3900.00);

INSERT INTO Passengers VALUES
(101, 'Arjun', 1, '2024-04-20'),
(102, 'Meena', 2, '2024-04-21'),
(103, 'Karan', 1, '2024-04-22'),
(104, 'Priya', 3, '2024-04-22'),
(105, 'Neeraj', 1, '2024-04-23');

-- 3. Passengers travelling to ‘Delhi’
SELECT p.name
FROM Passengers p
JOIN Flights f ON p.flight_id = f.flight_id
WHERE f.destination = 'Delhi';

-- 4. Flight-wise passenger count
SELECT f.flight_id, COUNT(p.pid) AS passenger_count
FROM Flights f
JOIN Passengers p ON f.flight_id = p.flight_id
GROUP BY f.flight_id;

-- 5. Increase fare by 10% for flights with more than 2 bookings
UPDATE Flights
SET fare = fare * 1.10
WHERE flight_id IN (
   SELECT flight_id
   FROM Passengers
   GROUP BY flight_id
   HAVING COUNT(*) > 2
);
```

Problem 5: Employee Performance Tracker
Schema:

- **Employee(emp_id INT, name VARCHAR(50), designation VARCHAR(30), salary INT)**
- **Performance(emp_id INT, month VARCHAR(15), rating INT)**

Questions:

1. Create schema and insert sample data.

2. Find employees with average rating > 4.

3. Display highest rated employee each month.

4. List employees who never received a rating using NOT IN.

5. Display total salary to be paid for 'Manager' designation employees.

```
-- Drop previous database if it exists
DROP DATABASE IF EXISTS performance_db;
CREATE DATABASE performance_db;
USE performance_db;

-- Drop existing tables if any
DROP TABLE IF EXISTS Performance;
DROP TABLE IF EXISTS Employee;

-- Create tables
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   name VARCHAR(50),
   designation VARCHAR(30),
   salary INT
);

CREATE TABLE Performance (
   emp_id INT,
   month VARCHAR(15),
   rating INT,
   FOREIGN KEY (emp_id) REFERENCES Employee(emp_id)
);

-- Insert sample data
INSERT INTO Employee VALUES
(1, 'Ravi Kumar', 'Manager', 45000),
(2, 'Seema Yadav', 'Developer', 28000),
(3, 'Ankit Mehta', 'Manager', 47000),
(4, 'Rohit Das', 'Tester', 26000),
(5, 'Preeti Sinha', 'Developer', 30000);

INSERT INTO Performance VALUES
(1, 'January', 5),
(2, 'January', 4),
(3, 'January', 3),
(4, 'January', 5),
(1, 'February', 4),
(2, 'February', 5),
(4, 'February', 4);

-- Query: Employees with average rating > 4
SELECT e.name, AVG(p.rating) AS avg_rating
FROM Employee e
JOIN Performance p ON e.emp_id = p.emp_id
GROUP BY e.emp_id
HAVING avg_rating > 4;

-- Query: Highest rated employee each month
SELECT p.month, e.name, p.rating
```

```
FROM Performance p
JOIN Employee e ON p.emp_id = e.emp_id
WHERE (p.month, p.rating) IN (
  SELECT month, MAX(rating)
  FROM Performance
  GROUP BY month
);

-- Query: Employees who never received a rating
SELECT name
FROM Employee
WHERE emp_id NOT IN (SELECT DISTINCT emp_id FROM Performance);

-- Query: Total salary of all 'Manager' designation employees
SELECT SUM(salary) AS total_salary
FROM Employee
WHERE designation = 'Manager';
```

Procedure:
A company wants to give a bonus of ₹5000 to employees whose salaries are less than ₹30,000. The HR department maintains a database of employee records.
Schema:
● Employees(emp_id INT PRIMARY KEY, name VARCHAR(50), salary INT, bonus INT DEFAULT 0)
Tasks:
1.Write a stored procedure using a cursor that:
● Retrieves all employees with salary < ₹30,000
● Adds ₹5000 to their bonus column
● Displays their name and updated bonus value

```
-- Drop procedure if exists
DROP PROCEDURE IF EXISTS update_bonus;

DELIMITER //
CREATE PROCEDURE update_bonus()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE empName VARCHAR(50);
  DECLARE empBonus INT;
  DECLARE empCursor CURSOR FOR
    SELECT name, bonus FROM Employees WHERE salary < 30000;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN empCursor;

  bonus_loop: LOOP
    FETCH empCursor INTO empName, empBonus;
    IF done THEN
      LEAVE bonus_loop;
    END IF;

    UPDATE Employees
    SET bonus = bonus + 5000
    WHERE name = empName;

    SELECT CONCAT('Bonus updated for ', empName, ': ₹', empBonus + 5000) AS Message;
```

```
    END LOOP;

    CLOSE empCursor;
END //
DELIMITER ;

-- Create sample table and data
DROP TABLE IF EXISTS Employees;

CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    salary INT,
    bonus INT DEFAULT 0
);

INSERT INTO Employees VALUES
(1, 'Ravi Kumar', 45000, 0),
(2, 'Seema Yadav', 28000, 0),
(3, 'Ankit Mehta', 47000, 0),
(4, 'Rohit Das', 26000, 0),
(5, 'Preeti Sinha', 30000, 0);

-- Call the procedure
CALL update_bonus();
```

2. A library tracks borrowed books and their return status. A fine of ₹2 is applied for each day after the due date.
Schema:
● Borrowers(borrow_id INT PRIMARY KEY, student_name VARCHAR(50), due_date DATE, return_date DATE, fine INT DEFAULT 0)
Task:
Write a stored procedure using a cursor to:
● Loop through all records in Borrowers
● For each student who returned the book late, calculate the number of overdue days
● Multiply overdue days by ₹2 and update the fine column
● Show a message like: Fine of ₹20 updated for Rahul Singh

```
-- Drop procedure if exists
DROP PROCEDURE IF EXISTS calculate_fine;

DELIMITER //
CREATE PROCEDURE calculate_fine()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE borrowId INT;
    DECLARE studentName VARCHAR(50);
    DECLARE due DATE;
    DECLARE returned DATE;
    DECLARE daysLate INT;

    DECLARE cur CURSOR FOR
        SELECT borrow_id, student_name, due_date, return_date FROM Borrowers;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;
```

```
    read_loop: LOOP
      FETCH cur INTO borrowId, studentName, due, returned;
      IF done THEN
        LEAVE read_loop;
      END IF;

      IF returned > due THEN
        SET daysLate = DATEDIFF(returned, due);
        UPDATE Borrowers
        SET fine = daysLate * 2
        WHERE borrow_id = borrowId;

        SELECT CONCAT('Fine of ₹', daysLate * 2, ' updated for ', studentName) AS Message;
      END IF;
    END LOOP;

    CLOSE cur;
END //
DELIMITER ;

-- Create sample table and data
DROP TABLE IF EXISTS Borrowers;

CREATE TABLE Borrowers (
    borrow_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    due_date DATE,
    return_date DATE,
    fine INT DEFAULT 0
);

INSERT INTO Borrowers VALUES
(1, 'Rahul Singh', '2025-04-15', '2025-04-20', 0),
(2, 'Neha Sharma', '2025-04-10', '2025-04-09', 0),
(3, 'Vikram Roy', '2025-04-12', '2025-04-17', 0);

-- Call the procedure
CALL calculate_fine();
```

Trigger
1: Track Salary Updates
Context:
A company wants to maintain a log of all salary changes for employees. Every time an employee's salary is updated, the old and new values should be stored in a separate table for audit purposes.
Tables:
● employees(emp_id INT PRIMARY KEY, name VARCHAR(50), salary DECIMAL(10,2))
● salary_log(log_id INT AUTO_INCREMENT PRIMARY KEY, emp_id INT, old_salary DECIMAL(10,2), new_salary DECIMAL(10,2), change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP)
Objective:
Create a BEFORE UPDATE trigger on the employees table that:
● Captures the old and new salary values whenever salary is updated
● Inserts them into the salary_log table

```
-- Drop existing tables if they exist
```

```
DROP TABLE IF EXISTS salary_log;
DROP TABLE IF EXISTS employees;

-- Create employees table
CREATE TABLE employees (
   emp_id INT PRIMARY KEY,
   name VARCHAR(50),
   salary DECIMAL(10,2)
);

-- Create salary_log table
CREATE TABLE salary_log (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   emp_id INT,
   old_salary DECIMAL(10,2),
   new_salary DECIMAL(10,2),
   change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create the trigger
DELIMITER //
CREATE TRIGGER before_salary_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
   IF OLD.salary != NEW.salary THEN
      INSERT INTO salary_log (emp_id, old_salary, new_salary)
      VALUES (OLD.emp_id, OLD.salary, NEW.salary);
   END IF;
END;
//
DELIMITER ;

-- Insert sample data
INSERT INTO employees VALUES
(1, 'Ravi Kumar', 45000.00),
(2, 'Seema Yadav', 28000.00),
(3, 'Ankit Mehta', 47000.00);

-- Sample update to trigger logging
UPDATE employees SET salary = 46000.00 WHERE emp_id = 1;

-- Check the salary log
SELECT * FROM salary_log;
```

Problem statements based on MongoDB
Student Performance Tracker
Context:
A university wants to track students' marks across various subjects using MongoDB.
Collection: students
Sample Document:
{
}
"roll_no": 101,
"name": "Ankita Desai",
"department": "IT",

"marks": [
{ "subject": "DBMS", "score": 78 },
{ "subject": "AI", "score": 89 },
{ "subject": "OS", "score": 91 }
]
Tasks:
1. Insert at least 5 student documents with varying subjects and marks.
2. Retrieve all students with more than 85 in "AI".
3. Update the DBMS score of student roll_no: 101 to 85.
4. Delete a student with roll_no: 105.
5. Use aggregation to find the average score in OS across all students.

```
// 1. Insert at least 5 student documents
db.students.insertMany([
 {
  roll_no: 101,
  name: "Ankita Desai",
  department: "IT",
  marks: [
   { subject: "DBMS", score: 78 },
   { subject: "AI", score: 89 },
   { subject: "OS", score: 91 }
  ]
 },
 {
  roll_no: 102,
  name: "Rohit Sharma",
  department: "CS",
  marks: [
   { subject: "DBMS", score: 85 },
   { subject: "AI", score: 92 },
   { subject: "OS", score: 88 }
  ]
 },
 {
  roll_no: 103,
  name: "Sneha Rane",
  department: "IT",
  marks: [
   { subject: "DBMS", score: 80 },
   { subject: "AI", score: 81 },
   { subject: "OS", score: 89 }
  ]
 },
 {
  roll_no: 104,
  name: "Yash Mehta",
  department: "CS",
  marks: [
   { subject: "DBMS", score: 90 },
   { subject: "AI", score: 86 },
   { subject: "OS", score: 85 }
  ]
 },
 {
  roll_no: 105,
  name: "Neha Kapoor",
```

```
  department: "ECE",
  marks: [
   { subject: "DBMS", score: 70 },
   { subject: "AI", score: 60 },
   { subject: "OS", score: 72 }
  ]
 }
]);

// 2. Retrieve students with more than 85 in AI
db.students.find({
 marks: { $elemMatch: { subject: "AI", score: { $gt: 85 } } }
});

// 3. Update DBMS score of student with roll_no 101 to 85
db.students.updateOne(
 { roll_no: 101, "marks.subject": "DBMS" },
 { $set: { "marks.$.score": 85 } }
);

// 4. Delete student with roll_no 105
db.students.deleteOne({ roll_no: 105 });

// 5. Use aggregation to find average score in OS
db.students.aggregate([
 { $unwind: "$marks" },
 { $match: { "marks.subject": "OS" } },
 {
  $group: {
   _id: null,
   avg_OS_score: { $avg: "$marks.score" }
  }
 }
]);
```

: Online Bookstore Database
Context:
An online bookstore wants to manage books, authors, and price data.
Collection: books
Sample Document:

```
{
"title": "The MongoDB Guide",
"author": "Ravi Joshi",
"price": 499,
"category": "Database",
"ratings": [4, 5, 5, 3]
}
```

Tasks:
1. Insert 5 books with details like title, author, price, category, and rating array.
2. Find all books priced under ₹500.
3. Update the price of a book titled "The MongoDB Guide" to ₹450.
4. Delete all books from category "Old Stock".
5. Use aggregation to calculate the average rating per book.

```
// 1. Insert 5 books
db.books.insertMany([
```

```
  {
    title: "The MongoDB Guide",
    author: "Ravi Joshi",
    price: 499,
    category: "Database",
    ratings: [4, 5, 5, 3]
  },
  {
    title: "Learn Python",
    author: "Neha Verma",
    price: 399,
    category: "Programming",
    ratings: [5, 5, 4]
  },
  {
    title: "AI for Beginners",
    author: "Suresh Gupta",
    price: 599,
    category: "AI",
    ratings: [4, 4, 3]
  },
  {
    title: "Data Structures",
    author: "Anita Sharma",
    price: 450,
    category: "Computer Science",
    ratings: [3, 4, 4]
  },
  {
    title: "Old MySQL Book",
    author: "Raj Patel",
    price: 300,
    category: "Old Stock",
    ratings: [2, 3]
  }
]);

// 2. Find all books priced under ₹500
db.books.find({ price: { $lt: 500 } });

// 3. Update the price of "The MongoDB Guide" to ₹450
db.books.updateOne(
  { title: "The MongoDB Guide" },
  { $set: { price: 450 } }
);

// 4. Delete all books from category "Old Stock"
db.books.deleteMany({ category: "Old Stock" });

// 5. Use aggregation to calculate average rating per book
db.books.aggregate([
  {
    $project: {
      title: 1,
      avg_rating: { $avg: "$ratings" }
    }
  }
```

]);

4: Hospital Patient Records System
Context:
A hospital wants to store and analyze basic patient treatment information.
Collection: patients
Sample Document
{
}
"patient_id": "P1001",
"name": "Rohan Kulkarni",
"age": 45,
"department": "Cardiology",
"treatments": [
{ "treatment": "ECG", "cost": 1200 },
{ "treatment": "Angiography", "cost": 15000 }
]
Tasks:
1. Insert 4–5 patient documents with multiple treatments.
2. Retrieve all patients from "Cardiology".
3. Add a new treatment for patient "P1001".
4. Delete records of patients older than 80 years.
5. Use aggregation to compute the total treatment cost per patient.

```
// 1. Insert 4–5 patient documents with multiple treatments
db.patients.insertMany([
 {
  patient_id: "P1001",
  name: "Rohan Kulkarni",
  age: 45,
  department: "Cardiology",
  treatments: [
   { treatment: "ECG", cost: 1200 },
   { treatment: "Angiography", cost: 15000 }
  ]
 },
 {
  patient_id: "P1002",
  name: "Suman Verma",
  age: 67,
  department: "Orthopedics",
  treatments: [
   { treatment: "X-Ray", cost: 800 },
   { treatment: "Fracture Repair", cost: 7000 }
  ]
 },
 {
  patient_id: "P1003",
  name: "Anil Sharma",
  age: 83,
  department: "Geriatrics",
  treatments: [
   { treatment: "Physiotherapy", cost: 2000 }
  ]
 },
 {
```

```
    patient_id: "P1004",
    name: "Neha Singh",
    age: 29,
    department: "Cardiology",
    treatments: [
      { treatment: "Stress Test", cost: 2500 },
      { treatment: "ECG", cost: 1200 }
    ]
  },
  {
    patient_id: "P1005",
    name: "Ravi Mehta",
    age: 52,
    department: "Neurology",
    treatments: [
      { treatment: "MRI", cost: 8000 }
    ]
  }
]);

// 2. Retrieve all patients from "Cardiology"
db.patients.find({ department: "Cardiology" });

// 3. Add a new treatment for patient "P1001"
db.patients.updateOne(
  { patient_id: "P1001" },
  { $push: { treatments: { treatment: "Echo", cost: 1800 } } }
);

// 4. Delete records of patients older than 80 years
db.patients.deleteMany({ age: { $gt: 80 } });

// 5. Aggregate total treatment cost per patient
db.patients.aggregate([
  { $unwind: "$treatments" },
  {
    $group: {
      _id: "$patient_id",
      name: { $first: "$name" },
      total_cost: { $sum: "$treatments.cost" }
    }
  }
]);
```

Problem 4: Movie Ratings and Reviews
Context:
A movie platform stores user reviews and wants to perform analysis on the data.
Collection: movies
Sample Document:

```
{
}
"movie_id": 1,
"title": "Interstellar",
"genre": "Sci-Fi",
"release_year": 2014,
"ratings": [
```

{ "user": "user1", "score": 5 },
{ "user": "user2", "score": 4 }
]
Tasks:
1. Insert at least 5 movie documents with ratings.
2. Find all movies released after 2010 in the "Sci-Fi" genre.
3. Update the title of a movie from "Inception" to "Inception (2010)".
4. Delete all movies with an average rating below 3.
5. Use aggregation to calculate the average score of each movie.

```
// 1. Insert at least 5 movie documents with ratings
db.movies.insertMany([
 {
  movie_id: 1,
  title: "Interstellar",
  genre: "Sci-Fi",
  release_year: 2014,
  ratings: [
   { user: "user1", score: 5 },
   { user: "user2", score: 4 }
  ]
 },
 {
  movie_id: 2,
  title: "Inception",
  genre: "Sci-Fi",
  release_year: 2010,
  ratings: [
   { user: "user1", score: 5 },
   { user: "user3", score: 4 }
  ]
 },
 {
  movie_id: 3,
  title: "Joker",
  genre: "Drama",
  release_year: 2019,
  ratings: [
   { user: "user2", score: 3 },
   { user: "user4", score: 4 }
  ]
 },
 {
  movie_id: 4,
  title: "Gravity",
  genre: "Sci-Fi",
  release_year: 2013,
  ratings: [
   { user: "user1", score: 2 },
   { user: "user2", score: 3 }
  ]
 },
 {
  movie_id: 5,
  title: "Toy Story",
  genre: "Animation",
  release_year: 1995,
```

```
    ratings: [
      { user: "user5", score: 5 },
      { user: "user6", score: 4 }
    ]
  }
]);

// 2. Find all movies released after 2010 in the "Sci-Fi" genre
db.movies.find({
  genre: "Sci-Fi",
  release_year: { $gt: 2010 }
});

// 3. Update the title of "Inception" to "Inception (2010)"
db.movies.updateOne(
  { title: "Inception" },
  { $set: { title: "Inception (2010)" } }
);

// 4. Delete all movies with average rating below 3
db.movies.aggregate([
  {
    $project: {
      _id: 1,
      avg_rating: { $avg: "$ratings.score" }
    }
  },
  {
    $match: {
      avg_rating: { $lt: 3 }
    }
  }
]).forEach(doc => db.movies.deleteOne({ _id: doc._id }));

// 5. Aggregate to calculate average score of each movie
db.movies.aggregate([
  {
    $project: {
      title: 1,
      avg_rating: { $avg: "$ratings.score" }
    }
  }
]);
```