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

```
create database finalPractical;
use finalPractical;

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,
    foreign key (projectid) references project(projectid),
    foreign key (eid) references employee(eid)
);

insert into employee (eid, ename, salary, address) values
(1, 'Alice', 45000, 'delhi'),
(2, 'Bob', 38000, 'Mumbai'),
(3, 'Charlie', 50000, 'Chennai'),
(4, 'David', 42000, 'Bangalore');
```

```
insert into manager (eid, ename) values
(10, 'Mr. Sharma'),
(11, 'Ms. Kapoor');

insert into project (projectid, project_name, manager) values
(101, 'Bank Management', 10),
(102, 'E-commerce Portal', 11),
(103, 'Hospital System', 10);

insert into assignment (projectid, eid) values
(101, 1),
(101, 2),
(102, 3),
(103, 4),
(102, 1);

-- 1. Alter table to add address in employee table
alter table employee add address varchar(100);

-- 2. Display employee name and projects on which they are working
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;

-- 3. Display projectid, projectname and their managers
select p.projectid, p.project_name, m.ename as manager_name
from project p
join manager m on p.manager = m.eid;

-- 4. Create view of employees working on 'Bank Management' project
create view bank_project_employess 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';

-- 5. Print names of employees whose salary is greater than 40000
select ename
from employee
where salary > 40000;

-- 6. Update salary of each employee with increase of Rs.2000
SET SQL_SAFE_UPDATES = 0;

update employee
set salary = salary + 2000;
```

```
SET SQL_SAFE_UPDATES = 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. 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.

```
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,
    FOREIGN KEY (projectid) REFERENCES project(projectid),
    FOREIGN KEY (eid) REFERENCES employee(eid)
);

INSERT INTO manager (eid, ename) VALUES
(101, 'Mr. Sharma'),
(102, 'Ms. Rani');

INSERT INTO project (projectid, project_name, manager) VALUES
(201, 'Bank Management', 101),
(202, 'Content Management', 102),
```

```
(203, 'E-commerce System', 101);

-- AUTO_INCREMENT will generate eid automatically
INSERT INTO employee (ename, salary) VALUES
('Alice', 45000),
('Bob', 39000),
('Charlie', 51000),
('David', 40000),
('Eva', 60000);

-- Alice (eid = 1) works on both Bank Management and Content Management
INSERT INTO assignment (projectid, eid) VALUES
(201, 1),
(202, 1),

-- Bob works only on Bank Management
(201, 2),

-- Charlie works only on Content Management
(202, 3),

-- David works only on E-commerce System
(203, 4),

-- Eva works on all three projects
(201, 5),
(202, 5),
(203, 5);

-- 1. Modify eid to use AUTO_INCREMENT
ALTER TABLE employee MODIFY eid INT AUTO_INCREMENT;

-- 2. Display employees working in both 'Bank Management' and 'Content Management'
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, e.ename
HAVING COUNT(DISTINCT p.project_name) = 2;

-- 3. Display average salary of organization
SELECT AVG(salary) AS avg_salary
FROM employee;

-- 4. Display employees who do not work on 'Bank Management' Project
SELECT e.name
FROM employee e
WHERE e.eid NOT IN (
        SELECT a.eid
```

```
    FROM assignment a
    JOIN project p ON a.projectid = p.projectid
    WHERE p.project_name = 'Bank Management'
);

-- 5. Delete employee whose id is 5
DELETE FROM employee
WHERE eid = 5;

-- 6. Display employee having highest salary in organization
SELECT ename, salary
FROM employee
WHERE salary = (SELECT MAX(salary) FROM employee);
```

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.

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

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

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
INSERT INTO supplier (supplierid, sname, saddress) VALUES
(1, 'Alpha Corp', 'Delhi'),
(2, 'Beta Ltd', 'Mumbai'),
(3, 'Gamma Inc', 'Chennai');

-- Insert into parts
INSERT INTO parts (part_id, part_name, color) VALUES
(1, 'Bolt', 'green'),
(2, 'Nut', 'blue'),
(3, 'Screw', 'red'),
(4, 'Washer', 'green');
```

```
-- Insert into catalog
INSERT INTO catalog (supplierid, part_id, cost) VALUES
(1, 1, 50),
(1, 2, 60),
(1, 3, 70),
(2, 1, 45),
(2, 2, 55),
(2, 4, 30),
(3, 1, 55),
(3, 2, 65),
(3, 3, 75),
(3, 4, 35);

-- 1. Find name of supplier who supply ‘green’ parts
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';

-- 2. Find name of suppliers who supply both blue and green parts
SELECT s.sname
FROM supplier s
WHERE s.supplierid IN (
    SELECT c1.supplierid
    FROM catalog c1
    JOIN parts p1 ON c1.part_id = p1.part_id
    WHERE p1.color = 'green'
)
AND s.supplierid IN (
    SELECT c2.supplierid
    FROM catalog c2
    JOIN parts p2 ON c2.part_id = p2.part_id
    WHERE p2.color = 'blue'
);

-- 3. Find supplier who supply all parts
SELECT s.sname
FROM supplier s
JOIN catalog c ON s.supplierid = c.supplierid
GROUP BY s.supplierid, s.sname
HAVING COUNT(DISTINCT c.part_id) = (SELECT COUNT(*) FROM parts);

-- 4. Find total cost of red parts
SELECT SUM(c.cost) AS total_red_part_cost
FROM catalog c
JOIN parts p ON c.part_id = p.part_id
WHERE p.color = 'red';
```

```sql
-- 5. Find supplier who supply green parts with minimum cost
SELECT s.sname, c.cost
FROM catalog c
JOIN parts p ON c.part_id = p.part_id
JOIN supplier s ON c.supplierid = s.supplierid
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'
);

-- 6. Update color of part having part_id = 4 and supplier_id = 2
UPDATE parts
SET color = 'yellow'
WHERE part_id = 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'.

```
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,
    FOREIGN KEY (eid) REFERENCES emp(eid),
    FOREIGN KEY (company_name) REFERENCES company(company_name)
);

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

INSERT INTO emp VALUES
(1, 'Prashant', 'MG Road', 'Pune'),
```

```
(2, 'Ravi', 'BTM Layout', 'Bangalore'),
(3, 'Sneha', 'Kothrud', 'Pune'),
(4, 'Arjun', 'Andheri', 'Mumbai');

INSERT INTO company VALUES
('Infosys', 'Bangalore'),
('TCS', 'Mumbai'),
('Accenture', 'Pune');

INSERT INTO works VALUES
(1, 'Infosys', 18000),
(2, 'TCS', 25000),
(3, 'TCS', 22000),
(4, 'Accenture', 30000);

INSERT INTO manages VALUES
(2, 1),
(3, 1),
(4, 2);

-- 1. Update company of employee name = ‘Prashant’ from ‘Infosys’ to ‘TCS’
UPDATE works
SET company_name = 'TCS'
WHERE eid = (SELECT eid FROM emp WHERE ename = 'Prashant')
        AND company_name = 'Infosys';

-- 2. Display names & cities of all employees who work for 'Infosys'
SELECT e.ename, e.city
FROM emp e
JOIN works w ON e.eid = w.eid
WHERE w.company_name = 'Infosys';

-- 3. Display names & street address of employees who work in TCS cities and earn
more than 20000
SELECT e.ename, e.street
FROM emp e
JOIN works w ON e.eid = w.eid
JOIN company c ON w.company_name = c.company_name
WHERE c.city = (SELECT city FROM company WHERE company_name = 'TCS')
        AND w.salary > 20000;

-- 4. Find all employees in database who do not work for 'Infosys'
SELECT e.ename
FROM emp e
JOIN works w ON e.eid = w.eid
WHERE w.company_name != 'Infosys';

-- 5. Find company-wise total salary
SELECT company_name, SUM(salary) AS total_salary
FROM works
```

```
GROUP BY company_name;

-- 6. Find names of all employees who work for 'Accenture'
SELECT e.ename
FROM emp e
JOIN works w ON e.eid = w.eid
WHERE w.company_name = 'Accenture';
```

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.

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

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

-- Insert into manager
INSERT INTO manager VALUES
(1001, 'Mr. Sharma'),
(1002, 'Ms. Rani');

-- Insert into project
```

```
INSERT INTO project VALUES
(201, 'Bank Management', 1001),
(202, 'Content Management', 1002),
(203, 'E-commerce', 1001);

-- Insert into employee
INSERT INTO employee (ename, salary) VALUES
('Alice', 50000),
('Bob', 42000),
('Charlie', 47000),
('David', 39000),
('Eva', 51000);

-- Insert into assignment
INSERT INTO assignment VALUES
(201, 1), -- Alice
(202, 1), -- Alice
(201, 2), -- Bob
(202, 3), -- Charlie
(203, 4), -- David
(201, 5), -- Eva
(202, 5); -- Eva

-- 1. Modify eid to use AUTO_INCREMENT
ALTER TABLE employee MODIFY eid INT AUTO_INCREMENT;

-- 2. Display employees working in both 'Bank Management' and 'Content Management'
SELECT e.ename
FROM employee e
WHERE e.eid IN (
    SELECT a1.eid
    FROM assignment a1
    JOIN project p1 ON a1.projectid = p1.projectid
    WHERE p1.project_name = 'Bank Management'
)
AND e.eid IN (
    SELECT a2.eid
    FROM assignment a2
    JOIN project p2 ON a2.projectid = p2.projectid
    WHERE p2.project_name = 'Content Management'
);

-- 3. Display average salary of organization
SELECT AVG(salary) AS avg_salary FROM employee;

-- 4. Display employees who do not work on 'Bank Management' Project
SELECT e.ename
FROM employee e
WHERE e.eid NOT IN (
    SELECT a.eid
```

```
    FROM assignment a
    JOIN project p ON a.projectid = p.projectid
    WHERE p.project_name = 'Bank Management'
);

-- 5. Delete employee whose id is 5
-- First delete from assignment
DELETE FROM assignment WHERE eid = 5;

-- Then delete from employee
DELETE FROM employee WHERE eid = 5;

-- 6. Display employee having highest salary in organization
SELECT ename, salary
FROM employee
WHERE salary = (SELECT  MAX(salary) FROM employee);
```

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.

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

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

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
INSERT INTO supplier (supplierid, sname, saddress) VALUES
(1, 'Alpha Corp', 'Delhi'),
(2, 'Beta Ltd', 'Mumbai'),
(3, 'Gamma Inc', 'Chennai');

-- Insert into parts
INSERT INTO parts (part_id, part_name, color) VALUES
(1, 'Bolt', 'green'),
(2, 'Nut', 'blue'),
(3, 'Screw', 'red'),
(4, 'Washer', 'green');
```

```
-- Insert into catalog
INSERT INTO catalog (supplierid, part_id, cost) VALUES
(1, 1, 50),
(1, 2, 60),
(1, 3, 70),
(2, 1, 45),
(2, 2, 55),
(2, 4, 30),
(3, 1, 55),
(3, 2, 65),
(3, 3, 75),
(3, 4, 35);

-- 1. Find name of supplier who supply ‘green’ parts
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';

-- 2. Find name of suppliers who supply both blue and green parts
SELECT s.sname
FROM supplier s
WHERE s.supplierid IN (
    SELECT c1.supplierid
    FROM catalog c1
    JOIN parts p1 ON c1.part_id = p1.part_id
    WHERE p1.color = 'green'
)
AND s.supplierid IN (
    SELECT c2.supplierid
    FROM catalog c2
    JOIN parts p2 ON c2.part_id = p2.part_id
    WHERE p2.color = 'blue'
);

-- 3. Find supplier who supply all parts
SELECT s.sname
FROM supplier s
JOIN catalog c ON s.supplierid = c.supplierid
GROUP BY s.supplierid, s.sname
HAVING COUNT(DISTINCT c.part_id) = (SELECT COUNT(*) FROM parts);

-- 4. Find total cost of red parts
SELECT SUM(c.cost) AS total_red_part_cost
FROM catalog c
JOIN parts p ON c.part_id = p.part_id
WHERE p.color = 'red';
```

```sql
-- 5. Find supplier who supply green parts with minimum cost
SELECT s.sname, c.cost
FROM catalog c
JOIN parts p ON c.part_id = p.part_id
JOIN supplier s ON c.supplierid = s.supplierid
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'
);

-- 6. Update color of part having part_id = 4 and supplier_id = 2
UPDATE parts
SET color = 'yellow'
WHERE part_id = 4;
```

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.

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

CREATE TABLE Cars (
    CarID INT PRIMARY KEY AUTO_INCREMENT,
    Model VARCHAR(50),
    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)
);

-- Customers
INSERT INTO Customers (Name, Email, Phone, City) VALUES
('Amit', 'amit@gmail.com', '9876543210', 'Delhi'),
('Sneha', 'sneha@gmail.com', '9876543211', 'Mumbai'),
('Rahul', 'rahul@gmail.com', '9876543212', 'Bangalore'),
('Priya', 'priya@gmail.com', '9876543213', 'Chennai');
```

```
-- Cars
INSERT INTO Cars (Model, Brand, Year, RentalPricePerDay, AvailabilityStatus) VALUES
('Swift', 'Maruti', 2021, 900, 'Available'),
('XUV500', 'Mahindra', 2022, 1200, 'Available'),
('City', 'Honda', 2023, 1100, 'Available'),
('Fortuner', 'Toyota', 2023, 1500, 'Available');

-- Rentals
INSERT INTO Rentals (CustomerID, CarID, StartDate, EndDate, TotalAmount) VALUES
(1, 2, '2024-03-01', '2024-03-05', 4800),  -- Amit rented Mahindra XUV500
(2, 3, '2024-03-10', '2024-03-15', 5500),  -- Sneha rented Honda City
(3, 4, '2024-03-20', '2024-03-25', 7500),  -- Rahul rented Toyota Fortuner
(4, 1, '2024-04-01', '2024-04-03', 1800);  -- Priya rented Swift

-- 1. Create a Payments table with attributes: PaymentID, RentalID (FK),
PaymentDate, AmountPaid, and PaymentMethod.
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)
);

-- Payments
INSERT INTO Payments (RentalID, PaymentDate, AmountPaid, PaymentMethod) VALUES
(1, '2024-03-05', 4800, 'Card'),
(2, '2024-03-15', 5500, 'UPI'),
(3, '2024-03-25', 7500, 'Cash'),
(4, '2024-04-03', 1800, 'Card');

-- 2. Update AvailabilityStatus of a car to 'Rented' for specific CustomerID and
CarID
UPDATE Cars
SET AvailabilityStatus = 'Rented'
WHERE CarID = (
        SELECT CarID FROM Rentals
    WHERE CustomerID = 1 AND CarID = 2
    LIMIT 1
);

-- 3. Retrieve Customer Name, Car Model, and Rental StartDate where
RentalPricePerDay > 1000
SELECT c.Name, ca.Model, r.StartDate
FROM Rentals r
JOIN Customers c ON r.CustomerID = c.CustomerID
JOIN Cars ca ON r.CarID = ca.CarID
WHERE ca.RentalPricePerDay > 1000;
```

```
-- 4. Calculate Total Rental Amount Collected per Car Brand
SELECT ca.Brand, SUM(r.TotalAmount) AS TotalRevenue
FROM Rentals r
JOIN Cars ca On r.CarID = ca.CarID
GROUP BY ca.Brand;

-- 5. Top 3 Customers Who Spent the Most on Rentals
SELECT cu.Name, SUM(r.TotalAmount) AS TotalSpent
FROM Rentals r
JOIN Customers cu ON r.CustomerID = r.CustomerID
GROUP BY cu.CustomerID
ORDER BY TotalSpent DESC
LIMIT 3;
```

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.

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Phone VARCHAR(15),
    Address VARCHAR(255)
);


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)
);

-- Customers
INSERT INTO Customers (Name, Email, Phone, Address) VALUES
('Amit', 'amit@example.com', '9876543210', 'Delhi'),
('Sneha', 'sneha@example.com', '9876543211', 'Mumbai'),
('Rahul', 'rahul@example.com', '9876543212', 'Bangalore'),
('Priya', 'priya@example.com', '9876543213', 'Chennai');

-- Products
INSERT INTO Products (Name, Category, Price, StockQuantity) VALUES
('iPhone 14', 'Electronics', 75000, 10),
('MacBook Air', 'Electronics', 95000, 5),
('T-Shirt', 'Clothing', 800, 50),
('Jeans', 'Clothing', 1500, 40),
('Coffee Mug', 'Kitchenware', 250, 100);

-- Orders
INSERT INTO Orders (CustomerID, OrderDate, TotalAmount) VALUES
(1, '2024-03-01', 150000),
(2, '2024-03-05', 6000),
(3, '2024-03-10', 1200),
(4, '2024-03-15', 300);

-- OrderDetails
INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Subtotal) VALUES
(1, 1, 1, 75000),
(1, 2, 1, 95000),
(2, 3, 5, 4000),
(2, 4, 2, 3000),
(3, 5, 4, 1000),
(4, 5, 1, 250);

-- 1. Create a Payments table with PaymentID, OrderID (FK), PaymentDate,
AmountPaid, and PaymentMethod.
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)
);

-- Payments
INSERT INTO Payments (OrderID, PaymentDate, AmountPaid, PaymentMethod) VALUES
(1, '2024-03-01', 150000, 'Card'),
```

```
(2, '2024-03-05', 6000, 'UPI'),
(3, '2024-03-10', 1200, 'Cash'),
(4, '2024-03-15', 300, 'Card');

-- 2. Update Stock Quantity of a Product After an Order is Placed
-- Reduce quantity for ProductID = 1 by 1 unit
UPDATE Products
SET StockQuantity = StockQuantity - 1
WHERE ProductID = 1;

-- 3. Retrieve Customer Name, Order Date, and TotalAmount where TotalAmount > 5000
SELECT c.Name, o.OrderDate, o.TotalAmount
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
WHERE o.TotalAmount > 5000;

-- 4. Calculate Total Sales per Product Category
SELECT p.Category, SUM(od.SubTotal) AS TotalSales
FROM OrderDetails od
JOIN Products p ON p.ProductID = od.ProductID
GROUP BY p.category;

-- 5. Top 5 Customers Who Spent the Most on Orders
SELECT c.Name, SUM(o.TotalAmount) AS TotalSpent
FROM Orders o
JOIN Customers c ON o.CustomerID = o.CustomerID
GROUP BY c.CustomerID
ORDER BY TotalSpent DESC
LIMIT 5;
```

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.

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

CREATE TABLE Books (
    BookID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(150),
    Author VARCHAR(100),
    Genre VARCHAR(50),
    CopiesAvailable INT
);

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)
);

-- Members
INSERT INTO Members (Name, Email, Phone, MembershipDate) VALUES
('Aarav Mehta', 'aarav@example.com', '9876543210', '2023-01-10'),
('Diya Sharma', 'diya@example.com', '9876543211', '2023-02-15'),
('Raj Patel', 'raj@example.com', '9876543212', '2023-03-20'),
```

```
('Sneha Reddy', 'sneha@example.com', '9876543213', '2023-04-25');

-- Books
INSERT INTO Books (Title, Author, Genre, CopiesAvailable) VALUES
('Wings of Fire', 'A.P.J. Abdul Kalam', 'Biography', 5),
('The Alchemist', 'Paulo Coelho', 'Fiction', 3),
('Python Programming', 'John Zelle', 'Education', 4),
('Ikigai', 'Francesc Miralles', 'Self-help', 2),
('Atomic Habits', 'James Clear', 'Self-help', 1);

-- BorrowedBooks
INSERT INTO BorrowedBooks (MemberID, BookID, BorrowDate, ReturnDate) VALUES
(1, 1, '2024-03-10', '2024-03-20'),
(2, 2, '2024-03-15', '2024-03-25'),
(3, 3, '2024-04-01', NULL),
(1, 4, '2024-04-05', NULL),
(4, 5, '2024-04-10', NULL);

-- 1. CREATE TABLE – Create a Fines table with FineID, MemberID (FK), Amount,
Status, and FineDate.
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)
);

-- Fines
INSERT INTO Fines (MemberID, Amount, Status, FineDate) VALUES
(1, 50.00, 'Unpaid', '2024-03-22'),
(2, 30.00, 'Paid', '2024-03-27');

-- 2. Update CopiesAvailable When a Book is Borrowed or Returned
-- Book borrowed (e.g., BookID = 3)
UPDATE Books
SET CopiesAvailable = CopiesAvailable - 1
WHERE BookID = 3;

-- Book returned (e.g., BookID = 3)
UPDATE Books
SET CopiesAvailable = CopiesAvailable + 1
WHERE BookID = 3;

-- 3. Retrieve Member Name, Book Title, and Borrow Date for books borrowed in the
last month
SELECT m.Name, b.Title, bb.BorrowDate
FROM BorrowedBooks bb
JOIN Members m ON bb.MemberID = m.MemberID
```

```
JOIN Books b ON bb.BookID = b.BookID
WHERE bb.BorrowDate >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH);

-- 4. Find the Number of Books Borrowed Per Genre
SELECT bk.Genre, COUNT(bb.BorrowID) AS BooksBorrowed
FROM BorrowedBooks bb
JOIN Books bk ON bb.BookID = bk.BookID
GROUP BY bk.Genre;

-- 5. Find the Top 5 Members Who Borrowed the Most Books
SELECT m.Name, COUNT(bb.BorrowID) AS BooksBorrowed
FROM BorrowedBooks bb
JOIN Members m ON bb.MemberID = m.MemberID
GROUP BY m.MemberID
ORDER BY BooksBorrowed DESC
LIMIT 5;
```

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.

```
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Contact VARCHAR(15)
);

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

CREATE TABLE Appointments (
    AppointmentID INT PRIMARY KEY AUTO_INCREMENT,
    PatientID INT,
    DoctorID INT,
    AppointmentDate DATE,
    Status VARCHAR(20),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);

CREATE TABLE Bills (
    BillID INT PRIMARY KEY AUTO_INCREMENT,
    PatientID INT,
    Amount DECIMAL(10,2),
```

```
    PaymentStatus VARCHAR(20),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

INSERT INTO Patients (Name, Age, Gender, Contact) VALUES
('Riya Sharma', 28, 'Female', '9876543210'),
('Aman Verma', 45, 'Male', '9876543211'),
('Kavita Desai', 32, 'Female', '9876543212'),
('Rohit Mehta', 51, 'Male', '9876543213');

INSERT INTO Doctors (Name, Specialization, Contact) VALUES
('Dr. Arjun Rao', 'Cardiology', '9123456780'),
('Dr. Meera Iyer', 'Dermatology', '9123456781'),
('Dr. Vivek Shah', 'Orthopedics', '9123456782');

INSERT INTO Appointments (PatientID, DoctorID, AppointmentDate, Status) VALUES
(1, 1, '2024-04-01', 'Scheduled'),
(2, 1, '2024-04-03', 'Completed'),
(3, 2, '2024-04-10', 'Scheduled'),
(4, 3, '2024-04-08', 'Completed'),
(1, 2, '2024-04-15', 'Scheduled');

INSERT INTO Bills (PatientID, Amount, PaymentStatus) VALUES
(1, 1500, 'Paid'),
(2, 2500, 'Paid'),
(3, 1800, 'Unpaid'),
(4, 2000, 'Paid');

-- 1. Create table – create a medicalrecords table with recordid, patientid (fk),
diagnosis, prescription, and recorddate.
CREATE TABLE MedicalRecords (
    RecordID INT PRIMARY KEY AUTO_INCREMENT,
    PatientID INT,
    Diagnosis TEXT,
    Prescription TEXT,
    RecordDate DATE,
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

INSERT INTO MedicalRecords (PatientID, Diagnosis, Prescription, RecordDate) VALUES
(2, 'High Blood Pressure', 'Amlodipine 5mg', '2024-04-03'),
(4, 'Knee Pain', 'Paracetamol 650mg', '2024-04-08');

--  2. Update an appointment status to "Completed"
UPDATE Appointments
SET Status = 'Completed'
WHERE AppointmentID = 1;

-- 3. Retrieve Patient Name, Doctor Name, and Appointment Date for patients who
consulted a specific specialization (e.g., 'Dermatology')
```

```sql
SELECT p.Name AS PatientName, d.Name AS DoctorName, a.AppointmentDate
FROM Appointments a
JOIN Patients p ON a.PatientID = p.PatientID
JOIN Doctors d ON a.DoctorID = d.DoctorID
WHERE d.Specialization = 'Dermatology';

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

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

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

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(50),
    Email VARCHAR(100)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY AUTO_INCREMENT,
    CourseName VARCHAR(100),
    Credits INT,
    Department VARCHAR(50)
);

CREATE TABLE Professors (
    ProfessorID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Department VARCHAR(50),
    Email VARCHAR(100)
);

CREATE TABLE Enrollments (
    StudentID INT,
    CourseID INT,
    Semester VARCHAR(20),
```

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

-- Students
INSERT INTO Students (Name, Age, Gender, Department, Email) VALUES
('Aman Singh', 20, 'Male', 'Computer Science', 'aman@uni.edu'),
('Riya Mehta', 22, 'Female', 'Electronics', 'riya@uni.edu'),
('Karan Patel', 21, 'Male', 'Mechanical', 'karan@uni.edu'),
('Priya Sharma', 23, 'Female', 'Computer Science', 'priya@uni.edu');

-- Courses
INSERT INTO Courses (CourseName, Credits, Department) VALUES
('Data Structures', 4, 'Computer Science'),
('Digital Circuits', 3, 'Electronics'),
('Thermodynamics', 4, 'Mechanical'),
('Algorithms', 4, 'Computer Science');

-- Professors
INSERT INTO Professors (Name, Department, Email) VALUES
('Dr. Verma', 'Computer Science', 'verma@uni.edu'),
('Dr. Iyer', 'Electronics', 'iyer@uni.edu'),
('Dr. Naik', 'Mechanical', 'naik@uni.edu');

-- Enrollments
INSERT INTO Enrollments (StudentID, CourseID, Semester, EnrollmentDate, Grade)
VALUES
(1, 1, 'Spring2025', '2025-01-10', 'A'),
(1, 4, 'Spring2025', '2025-01-11', 'B'),
(2, 2, 'Spring2025', '2025-01-12', 'A'),
(3, 3, 'Spring2025', '2025-01-13', 'B+'),
(4, 1, 'Spring2025', '2025-01-14', 'A'),
(4, 4, 'Spring2025', '2025-01-15', 'A');

-- 1. Percentage of Students in Each Department
SELECT Department, ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM Students), 2) AS
Percentage
FROM Students
GROUP BY Department;

-- 2. Detect Duplicate Enrollments
SELECT StudentID, CourseID, Semester, COUNT(*) AS Count
FROM Enrollments
GROUP BY StudentID, CourseID, Semester
HAVING COUNT(*) > 1;

-- 3. Semester with Highest Average Enrollments per Course
```

```
SELECT Semester, ROUND(COUNT(*) * 1.0 / (SELECT COUNT(*) FROM Courses), 2) AS
AvgEnrollmentsPerCourse
FROM Enrollments
GROUP BY Semester
ORDER BY AvgEnrollmentsPerCourse DESC
LIMIT 1;

-- 4. Students with More Than 3 Enrollments
SELECT s.Name, COUNT(e.CourseID) AS TotalEnrollments
FROM Enrollments e
JOIN Students s ON e.StudentId = s.StudentID
GROUP BY e.StudentID
HAVING COUNT(e.CourseID) > 3;

-- 5. Courses and Number of Students Enrolled
SELECT c.CourseName, COUNT(e.StudentID) AS NumberOfStudents
FROM Courses c
LEFT JOIN Enrollments e ON c.CourseID = e.CourseId
GROUP BY c.CourseID, c.CourseName;
```

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)

```
-- Student Management
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(50),
    Email VARCHAR(100)
);

-- Course Management
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY AUTO_INCREMENT,
    CourseName VARCHAR(100),
    Credits INT,
    Department VARCHAR(50)
);

-- Enrollment System
CREATE TABLE Enrollments (
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    Grade FLOAT,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

```sql
-- Professor Management
CREATE TABLE Professors (
    ProfessorID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Department VARCHAR(50),
    Email VARCHAR(100)
);

-- Students
INSERT INTO Students (Name, Age, Gender, Department, Email) VALUES
('Pooja', 21, 'Female', 'Computer Science', 'pooja@example.com'),
('Ravi', 22, 'Male', 'Electronics', 'ravi@example.com'),
('Meena', 20, 'Female', 'Mechanical', 'meena@example.com'),
('Amit', 23, 'Male', 'Computer Science', 'amit@example.com');

-- Courses
INSERT INTO Courses (CourseName, Credits, Department) VALUES
('Data Structures', 4, 'Computer Science'),
('Digital Circuits', 3, 'Electronics'),
('Thermodynamics', 4, 'Mechanical'),
('Algorithms', 4, 'Computer Science'),
('Machine Learning', 3, 'Computer Science');

-- Enrollments
INSERT INTO Enrollments (StudentID, CourseID, EnrollmentDate, Grade) VALUES
(1, 1, '2025-01-10', 3.5),
(1, 4, '2025-01-12', 2.7),
(2, 2, '2025-01-15', 1.8),
(2, 1, '2025-01-16', 1.5),
(2, 4, '2025-01-17', 1.0),
(2, 5, '2025-01-18', 2.8),
(3, 3, '2025-01-11', 3.0);

INSERT INTO Professors (Name, Department, Email) VALUES
('Dr. Sharma', 'Computer Science', 'sharma.cs@university.edu'),
('Dr. Reddy', 'Electronics', 'reddy.ec@university.edu'),
('Dr. Verma', 'Mechanical', 'verma.me@university.edu'),
('Dr. Khan', 'Computer Science', 'khan.cs@university.edu');

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

--  2. Identify students who failed more than 2 courses (grade < 2.0)
SELECT s.Name, COUNT(*) AS FailedCourses
FROM Enrollments e
```

```
JOIN Students s ON e.StudentID = s.StudentID
WHERE e.Grade < 2.0
GROUP BY s.StudentID
HAVING COUNT(*) > 2;

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

-- 4. Find courses with zero enrollments
SELECT CourseName
FROM Courses
WHERE CourseID NOT IN (
    SELECT DISTINCT CourseID FROM Enrollments
);

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

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

```
-- Students
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(50),
    Email VARCHAR(100)
);

-- Courses
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY AUTO_INCREMENT,
    CourseName VARCHAR(100),
    Credits INT,
    Department VARCHAR(50)
);

-- Enrollments
CREATE TABLE Enrollments (
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    Grade FLOAT,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

```
-- Professors
CREATE TABLE Professors (
    ProfessorID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Department VARCHAR(50),
    Email VARCHAR(100)
);

-- Students
INSERT INTO Students (Name, Age, Gender, Department, Email) VALUES
('Pooja', 21, 'Female', 'Computer Science', 'pooja@example.com'),
('Ravi', 22, 'Male', 'Electronics', 'ravi@example.com'),
('Meena', 20, 'Female', 'Mechanical', 'meena@example.com'),
('Amit', 23, 'Male', 'Computer Science', 'amit@example.com'),
('Neha', 22, 'Female', 'Computer Science', 'neha@example.com');

-- Courses
INSERT INTO Courses (CourseName, Credits, Department) VALUES
('Data Structures', 4, 'Computer Science'),
('Digital Circuits', 3, 'Electronics'),
('Thermodynamics', 4, 'Mechanical'),
('Algorithms', 4, 'Computer Science'),
('Machine Learning', 3, 'Computer Science');

-- Enrollments
INSERT INTO Enrollments (StudentID, CourseID, EnrollmentDate, Grade) VALUES
(1, 1, '2025-01-10', 3.5),
(1, 4, '2025-01-12', 2.7),
(1, 5, '2025-01-14', 3.2),
(2, 2, '2025-01-15', 1.8),
(2, 1, '2025-01-16', 1.5),
(2, 4, '2025-01-17', 1.0),
(2, 5, '2025-01-18', 2.8),
(3, 3, '2025-01-11', 3.0),
(4, 1, '2025-01-10', 2.9),
(4, 4, '2025-01-12', 3.7),
(4, 5, '2025-01-14', 3.1),
(4, 2, '2025-01-15', 2.5);

INSERT INTO Professors (Name, Department, Email) VALUES
('Dr. Sharma', 'Computer Science', 'sharma@university.edu'),
('Dr. Iyer', 'Electronics', 'iyer@university.edu'),
('Dr. Rao', 'Mechanical', 'rao@university.edu'),
('Dr. Verma', 'Computer Science', 'verma@university.edu'),
('Dr. Joshi', 'Mathematics', 'joshi@university.edu');

-- 1. Find students who have not enrolled in any course
SELECT s.StudentID, s.Name
FROM Students s
```

```
WHERE s.StudentID NOT IN (
    SELECT DISTINCT StudentID FROM Enrollments
);

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

-- 3. Find the average grade of students per course
SELECT c.CourseName, ROUND(AVG(e.Grade), 2) AS AvgGrade
FROM Enrollments e
JOIN Courses c ON e.CourseID = c.CourseID
GROUP BY e.CourseID;

-- 4. Retrieve the highest grade in each course
SELECT c.CourseName, MAX(e.Grade) AS HighestGrade
FROM Enrollments e
JOIN Courses c ON e.CourseID = c.CourseID
GROUP BY e.CourseID;

-- 5. 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;
```

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)

```
CREATE TABLE Customer (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    address VARCHAR(255),
    phone VARCHAR(15),
    email VARCHAR(100)
);

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

CREATE TABLE Employee (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    position VARCHAR(100),
    branch_id INT,
    salary DECIMAL(10,2),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);

CREATE TABLE Account (
    account_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    account_type VARCHAR(50),
    balance DECIMAL(12,2),
    branch_id INT,
```

```
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);

CREATE TABLE Transaction (
    transaction_id INT PRIMARY KEY AUTO_INCREMENT,
    account_id INT,
    transaction_type VARCHAR(50),
    amount DECIMAL(10,2),
    transaction_date DATE,
    FOREIGN KEY (account_id) REFERENCES Account(account_id)
);

CREATE TABLE Loan (
    loan_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    amount DECIMAL(12,2),
    loan_type VARCHAR(50),
    status VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

-- Customers
INSERT INTO Customer (name, address, phone, email) VALUES
('Rahul', 'Delhi', '9876543210', 'rahul@gmail.com'),
('Sneha', 'Mumbai', '9123456780', 'sneha@gmail.com'),
('Amit', 'Bangalore', '9988776655', 'amit@gmail.com');

-- Branches
INSERT INTO Branch (branch_name, location, manager_id) VALUES
('Main Branch', 'Delhi', 1),
('City Branch', 'Mumbai', 2);

-- Employees
INSERT INTO Employee (name, position, branch_id, salary) VALUES
('Rakesh', 'Manager', 1, 80000),
('Priya', 'Clerk', 2, 40000);

-- Accounts
INSERT INTO Account (customer_id, account_type, balance, branch_id) VALUES
(1, 'Savings', 120000.00, 1),
(2, 'Current', 95000.00, 2),
(3, 'Savings', 180000.00, 1);

-- Transactions
INSERT INTO Transaction (account_id, transaction_type, amount, transaction_date)
VALUES
(1, 'Deposit', 5000.00, '2024-01-15'),
(1, 'Withdrawal', 2000.00, '2024-02-10'),
(2, 'Deposit', 7000.00, '2024-03-01');
```

```sql
-- Loans
INSERT INTO Loan (customer_id, amount, loan_type, status) VALUES
(1, 150000.00, 'Home', 'Approved'),
(2, 80000.00, 'Personal', 'Pending'),
(3, 200000.00, 'Car', 'Approved');

-- 1. List all customers and their account details
SELECT c.name, c.phone, a.account_id, a.account_type, a.balance
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.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: 1)
SELECT * FROM Transaction
WHERE account_id = 1;

-- 5. Find customers who have both a loan and an account
SELECT 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.name, a.account_id, a.balance
FROM Customer c
JOIN Account a ON c.customer_id = a.customer_id
WHERE a.balance > 100000;

SELECT * FROM HighValueCustomers;
```

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

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

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

CREATE TABLE Account (
    account_id INT PRIMARY KEY,
    customer_id INT,
    account_type VARCHAR(20),
    balance DECIMAL(10,2),
    branch_id INT,
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);

CREATE TABLE Transaction (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    transaction_type VARCHAR(20),
    amount DECIMAL(10,2),
```

```
    transaction_date DATE,
    FOREIGN KEY (account_id) REFERENCES Account(account_id)
);

CREATE TABLE Loan (
    loan_id INT PRIMARY KEY,
    customer_id INT,
    amount DECIMAL(10,2),
    loan_type VARCHAR(50),
    status VARCHAR(20),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

CREATE TABLE Employee (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    position VARCHAR(50),
    branch_id INT,
    salary DECIMAL(10,2),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);

-- Customers
INSERT INTO Customer VALUES
(1, 'Ravi', 'Mumbai', '9876543210', 'ravi@gmail.com'),
(2, 'Anita', 'Delhi', '9123456789', 'anita@gmail.com');

-- Branches
INSERT INTO Branch VALUES
(1, 'Main', 'Mumbai', 101),
(2, 'South', 'Delhi', 102),
(3, 'North', 'Chennai', 103);

-- Accounts
INSERT INTO Account VALUES
(101, 1, 'Savings', 8000, 1),
(102, 2, 'Current', 4000, 3),
(105, 1, 'Savings', 10000, 3);

-- Transactions
INSERT INTO Transaction VALUES
(1, 101, 'Deposit', 3000, '2024-12-01'),
(2, 102, 'Withdrawal', 500, '2024-12-03'),
(3, 101, 'Deposit', 20000, '2024-12-05');

-- Loans
INSERT INTO Loan VALUES
(1, 1, 150000, 'Home Loan', 'Approved'),
(2, 2, 30000, 'Personal Loan', 'Rejected');
```

```
-- Employees
INSERT INTO Employee VALUES
(101, 'Manoj', 'Manager', 1, 50000),
(102, 'Sita', 'Clerk', 3, 20000),
(103, 'Arjun', 'Assistant', 3, 18000);

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

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

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

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

-- Delete inactive loan applications (status = 'Rejected')
SET SQL_SAFE_UPDATES = 0;

DELETE FROM Loan
WHERE status = 'Rejected';

SET SQL_SAFE_UPDATES = 1;

-- Calculate the total loan amount per loan type
SELECT loan_type, SUM(amount) AS total_amount
FROM Loan
GROUP BY loan_type;
```