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

```
-- Create Tables
CREATE TABLE employee (
    eid INT PRIMARY KEY,
    ename VARCHAR(255),
    salary DECIMAL(10,2)
);

CREATE TABLE project (
    projectid INT PRIMARY KEY,
    project_name VARCHAR(255),
    manager INT,
    FOREIGN KEY (manager) REFERENCES employee(eid)
);

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

CREATE TABLE manager (
    eid INT PRIMARY KEY,
    ename VARCHAR(255),
    FOREIGN KEY (eid) REFERENCES employee(eid)
);
```

```
-- 1. Alter table to add address in employee table
ALTER TABLE employee ADD COLUMN address VARCHAR(255);

-- 2. Display employee name and projects on which they are working
SELECT employee.ename, project.project_name
FROM employee
JOIN assignment ON employee.eid = assignment.eid
JOIN project ON assignment.projectid = project.projectid;

-- 3. Display projectid, projectname and their managers
SELECT project.projectid, project.project_name, manager.ename AS manager_name
FROM project
JOIN manager ON project.manager = manager.eid;

-- 4. Create view of employees working on 'Bank Management' project
CREATE VIEW BankManagementEmployees AS
SELECT employee.ename
FROM employee
JOIN assignment ON employee.eid = assignment.eid
JOIN project ON assignment.projectid = project.projectid
WHERE project.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
UPDATE employee
SET salary = salary + 2000;
```

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:

```
-- Create Tables

CREATE TABLE employee (
    eid INT PRIMARY KEY AUTO_INCREMENT,
    ename VARCHAR(255),
    salary DECIMAL(10,2)
);


CREATE TABLE project (
    projectid INT PRIMARY KEY,
    project_name VARCHAR(255),
    manager INT,
    FOREIGN KEY (manager) REFERENCES employee(eid)
);


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


CREATE TABLE manager (
    eid INT PRIMARY KEY,
    ename VARCHAR(255),
    FOREIGN KEY (eid) REFERENCES employee(eid)
);
```

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

-- 2. Display Employees working in both projects 'Bank Management' and 'Content Management'
SELECT employee.ename
FROM employee
JOIN assignment AS assignment1 ON employee.eid = assignment1.eid
JOIN project AS project1 ON assignment1.projectid = project1.projectid
JOIN assignment AS assignment2 ON employee.eid = assignment2.eid
JOIN project AS project2 ON assignment2.projectid = project2.projectid
WHERE project1.project_name = 'Bank Management'
AND project2.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 employee.ename
FROM employee
WHERE employee.eid NOT IN (
   SELECT assignment.eid
   FROM assignment
   JOIN project ON assignment.projectid = project.projectid
   WHERE project.project_name = 'Bank Management'
);

-- 5. Delete Employee Whose ID is 5
DELETE FROM employee
WHERE eid = 5;
```

-- 6. Display Employee Having the Highest Salary in Organization

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

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

```
-- Create Tables
CREATE TABLE supplier (
   supplierid INT PRIMARY KEY,
   sname VARCHAR(255),
   saddress VARCHAR(255)
);

CREATE TABLE parts (
   part_id INT PRIMARY KEY,
   part_name VARCHAR(255),
   color VARCHAR(50)
);

CREATE TABLE catalog (
   supplierid INT,
   part_id INT,
   cost DECIMAL(10,2),
   PRIMARY KEY (supplierid, part_id),
   FOREIGN KEY (supplierid) REFERENCES supplier(supplierid),
   FOREIGN KEY (part_id) REFERENCES parts(part_id)
);
```

```
-- 1. Find name of supplier who supplies ‘green’ parts
SELECT DISTINCT supplier.sname
FROM supplier
JOIN catalog ON supplier.supplierid = catalog.supplierid
JOIN parts ON catalog.part_id = parts.part_id
WHERE parts.color = 'green';

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

-- 3. Find suppliers who supply all parts
SELECT supplier.sname
FROM supplier
WHERE NOT EXISTS (
   SELECT parts.part_id
   FROM parts
   WHERE NOT EXISTS (
      SELECT catalog.part_id
      FROM catalog
      WHERE catalog.supplierid = supplier.supplierid
      AND catalog.part_id = parts.part_id
   )
);

-- 4. Find total cost of red parts
```

```
SELECT SUM(catalog.cost) AS total_cost
FROM catalog
JOIN parts ON catalog.part_id = parts.part_id
WHERE parts.color = 'red';

-- 5. Find supplier who supplies green parts with minimum cost
SELECT supplier.sname
FROM supplier
JOIN catalog ON supplier.supplierid = catalog.supplierid
JOIN parts ON catalog.part_id = parts.part_id
WHERE parts.color = 'green'
ORDER BY catalog.cost ASC
LIMIT 1;

-- 6. Update color of part having part_id = 4 and supplier_id = 2
UPDATE parts
SET color = 'new_color'
WHERE part_id = 4
AND part_id IN (
   SELECT part_id
   FROM catalog
   WHERE supplierid = 2
);
```

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 Tables
```

```
CREATE TABLE emp (
    eid INT PRIMARY KEY,
    ename VARCHAR(255),
    street VARCHAR(255),
    city VARCHAR(255)
);

CREATE TABLE company (
    company_name VARCHAR(255) PRIMARY KEY,
    city VARCHAR(255)
);

CREATE TABLE works (
    eid INT,
    company_name VARCHAR(255),
    salary DECIMAL(10,2),
    PRIMARY KEY (eid, company_name),
    FOREIGN KEY (eid) REFERENCES emp(eid),
    FOREIGN KEY (company_name) REFERENCES company(company_name)
);

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

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

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

-- 4. Find all employees in the database who do not work for 'Infosys'
SELECT emp.ename
FROM emp
WHERE emp.eid NOT IN (
   SELECT works.eid FROM works WHERE works.company_name = 'Infosys'
);

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

-- 6. Find names of all employees who work for 'Accenture'
SELECT emp.ename
```

```
FROM emp
JOIN works ON emp.eid = works.eid
WHERE works.company_name = 'Accenture';
```

5 . 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 Tables
CREATE TABLE employee (
   eid INT PRIMARY KEY AUTO_INCREMENT,
   ename VARCHAR(255),
   salary DECIMAL(10,2)
);

CREATE TABLE project (
   projectid INT PRIMARY KEY,
   project_name VARCHAR(255),
   manager INT,
   FOREIGN KEY (manager) REFERENCES employee(eid)
);

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

```
CREATE TABLE manager (
   eid INT PRIMARY KEY,
   ename VARCHAR(255),
   FOREIGN KEY (eid) REFERENCES employee(eid)
);

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

-- 2. Display Employees working in both projects 'Bank Management' and 'Content Management'
SELECT employee.ename
FROM employee
JOIN assignment AS assignment1 ON employee.eid = assignment1.eid
JOIN project AS project1 ON assignment1.projectid = project1.projectid
JOIN assignment AS assignment2 ON employee.eid = assignment2.eid
JOIN project AS project2 ON assignment2.projectid = project2.projectid
WHERE project1.project_name = 'Bank Management'
AND project2.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 employee.ename
FROM employee
WHERE employee.eid NOT IN (
   SELECT assignment.eid
   FROM assignment
   JOIN project ON assignment.projectid = project.projectid
```

```
    WHERE project.project_name = 'Bank Management'
);

-- 5. Delete Employee Whose ID is 5
DELETE FROM employee
WHERE eid = 5;

-- 6. Display Employee Having the Highest Salary in Organization
SELECT ename
FROM employee
WHERE salary = (SELECT MAX(salary) FROM employee);
```

6. 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 Tables
CREATE TABLE supplier (
    supplierid INT PRIMARY KEY,
    sname VARCHAR(255),
    saddress VARCHAR(255)
);

CREATE TABLE parts (
    part_id INT PRIMARY KEY,
    part_name VARCHAR(255),
    color VARCHAR(50)
);

CREATE TABLE catalog (
```

```
    supplierid INT,
    part_id INT,
    cost DECIMAL(10,2),
    PRIMARY KEY (supplierid, part_id),
    FOREIGN KEY (supplierid) REFERENCES supplier(supplierid),
    FOREIGN KEY (part_id) REFERENCES parts(part_id)
);

-- 1. Find name of supplier who supplies ‘green’ parts
SELECT DISTINCT supplier.sname
FROM supplier
JOIN catalog ON supplier.supplierid = catalog.supplierid
JOIN parts ON catalog.part_id = parts.part_id
WHERE parts.color = 'green';

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

-- 3. Find suppliers who supply all parts
SELECT supplier.sname
```

```
FROM supplier
WHERE NOT EXISTS (
   SELECT parts.part_id FROM parts
   WHERE NOT EXISTS (
      SELECT catalog.part_id FROM catalog
      WHERE catalog.supplierid = supplier.supplierid
      AND catalog.part_id = parts.part_id
   )
);

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

-- 5. Find supplier who supplies green parts with minimum cost
SELECT supplier.sname
FROM supplier
JOIN catalog ON supplier.supplierid = catalog.supplierid
JOIN parts ON catalog.part_id = parts.part_id
WHERE parts.color = 'green'
ORDER BY catalog.cost ASC
LIMIT 1;

-- 6. Update color of part having part_id = 4 and supplier_id = 2
UPDATE parts
SET color = 'new_color'
WHERE part_id = 4
AND part_id IN (
   SELECT part_id FROM catalog WHERE supplierid = 2
```

);

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.

```
-- Create Tables
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Phone VARCHAR(20),
    City VARCHAR(100)
);

CREATE TABLE Cars (
    CarID INT PRIMARY KEY,
    Model VARCHAR(255),
    Brand VARCHAR(255),
    Year INT,
    RentalPricePerDay DECIMAL(10,2),
    AvailabilityStatus VARCHAR(50)
);

CREATE TABLE Rentals (
    RentalID INT PRIMARY KEY,
    CustomerID INT,
    CarID INT,
```

```
    StartDate DATE,
    EndDate DATE,
    TotalAmount DECIMAL(10,2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (CarID) REFERENCES Cars(CarID)
);

-- 1. Create a Payments Table
CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY,
    RentalID INT,
    PaymentDate DATE,
    AmountPaid DECIMAL(10,2),
    PaymentMethod VARCHAR(50),
    FOREIGN KEY (RentalID) REFERENCES Rentals(RentalID)
);

-- 2. Update AvailabilityStatus of a Car to 'Rented' for a Specific CustomerID and CarID
UPDATE Cars
SET AvailabilityStatus = 'Rented'
WHERE CarID = (SELECT CarID FROM Rentals WHERE CustomerID = X);

-- 3. Retrieve Customer Name, Car Model, and Rental StartDate for Rentals Where RentalPricePerDay
is Above 1000
SELECT Customers.Name, Cars.Model, Rentals.StartDate
FROM Customers
JOIN Rentals ON Customers.CustomerID = Rentals.CustomerID
JOIN Cars ON Rentals.CarID = Cars.CarID
WHERE Cars.RentalPricePerDay > 1000;

-- 4. Calculate the Total Rental Amount Collected Per Car Brand
```

```
SELECT Cars.Brand, SUM(Rentals.TotalAmount) AS TotalRevenue
FROM Cars
JOIN Rentals ON Cars.CarID = Rentals.CarID
GROUP BY Cars.Brand;

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

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.

```
-- Create Tables
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Phone VARCHAR(20),
    Address VARCHAR(255)
);

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    Name VARCHAR(255),
```

```
    Category VARCHAR(100),
    Price DECIMAL(10,2),
    StockQuantity INT
);

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

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

-- 1. Create a Payments Table
CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY,
    OrderID INT,
    PaymentDate DATE,
    AmountPaid DECIMAL(10,2),
    PaymentMethod VARCHAR(50),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
```

```
);

-- 2. Update the Stock Quantity of a Product After an Order is Placed
UPDATE Products
SET StockQuantity = StockQuantity - (
    SELECT Quantity FROM OrderDetails WHERE ProductID = X AND OrderID = Y
)
WHERE ProductID = X;

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

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

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

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

```
-- Create Tables
CREATE TABLE Members (
    MemberID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Phone VARCHAR(20),
    MembershipDate DATE
);

CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(255),
    Author VARCHAR(255),
    Genre VARCHAR(100),
    CopiesAvailable INT
);

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

```
-- 1. Create a Fines Table
CREATE TABLE Fines (
    FineID INT PRIMARY KEY,
    MemberID INT,
    Amount DECIMAL(10,2),
    Status VARCHAR(50),
    FineDate DATE,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);

-- 2. Update CopiesAvailable When a Book is Borrowed or Returned
UPDATE Books
SET CopiesAvailable = CopiesAvailable - 1
WHERE BookID = 1; -- When Borrowed

UPDATE Books
SET CopiesAvailable = CopiesAvailable + 1
WHERE BookID = 1; -- When Returned

-- 3. Retrieve Member Name, Book Title, and Borrow Date for Books Borrowed in the Last Month
SELECT m.Name, b.Title, bb.BorrowDate
FROM Members m
JOIN BorrowedBooks bb ON m.MemberID = bb.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 b.Genre, COUNT(bb.BorrowID) AS BooksBorrowed
FROM Books b
JOIN BorrowedBooks bb ON b.BookID = bb.BookID
```

```
GROUP BY b.Genre;


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

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.

```
-- Create Tables
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY,
    Name VARCHAR(255),
    Age INT,
    Gender VARCHAR(10),
    Contact VARCHAR(50)
);


CREATE TABLE Doctors (
    DoctorID INT PRIMARY KEY,
    Name VARCHAR(255),
    Specialization VARCHAR(100),
    Contact VARCHAR(50)
```

```
);

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 TABLE Bills (
    BillID INT PRIMARY KEY,
    PatientID INT,
    Amount DECIMAL(10,2),
    PaymentStatus VARCHAR(50),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

-- 1. Create a MedicalRecords Table
CREATE TABLE MedicalRecords (
    RecordID INT PRIMARY KEY,
    PatientID INT,
    Diagnosis TEXT,
    Prescription TEXT,
    RecordDate DATE,
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

-- 2. Update Appointment Status to "Completed" After a Patient’s Visit
```

```
UPDATE Appointments
SET Status = 'Completed'
WHERE AppointmentID = 1;

-- 3. Retrieve Patient Name, Doctor Name, and Appointment Date for Patients Who Consulted a Specific Specialization
SELECT p.Name AS PatientName, d.Name AS DoctorName, a.AppointmentDate
FROM Patients p
JOIN Appointments a ON p.PatientID = a.PatientID
JOIN Doctors d ON a.DoctorID = d.DoctorID
WHERE d.Specialization = 'Cardiology';  -- Change 'Cardiology' to required specialization

-- 4. Find the Total Revenue Collected Per Doctor
SELECT d.Name AS DoctorName, SUM(b.Amount) AS TotalRevenue
FROM Doctors d
JOIN Appointments a ON d.DoctorID = a.DoctorID
JOIN Bills b ON a.PatientID = b.PatientID
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 TotalAppointments
FROM Doctors d
JOIN Appointments a ON d.DoctorID = a.DoctorID
GROUP BY d.DoctorID
ORDER BY TotalAppointments DESC
LIMIT 3;
```

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

```
-- Create Tables
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(255),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(100),
    Email VARCHAR(255)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(255),
    Credits INT,
    Department VARCHAR(100)
);

CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY AUTO_INCREMENT,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    Grade DECIMAL(3,2),
    Semester VARCHAR(20),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

```
CREATE TABLE Professors (
    ProfessorID INT PRIMARY KEY,
    Name VARCHAR(255),
    Department VARCHAR(100),
    Email VARCHAR(255)
);

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

-- 2. Detect Duplicate Enrollments (Same Student Enrolled in the Same Course in the Same Semester)
SELECT StudentID, CourseID, Semester, COUNT(*) AS DuplicateCount
FROM Enrollment
GROUP BY StudentID, CourseID, Semester
HAVING COUNT(*) > 1;

-- 3. Find the Semester with the Highest Average Enrollments Per Course
SELECT Semester, AVG(TotalEnrollments) AS AvgEnrollments
FROM (
    SELECT Semester, CourseID, COUNT(StudentID) AS TotalEnrollments
    FROM Enrollment
    GROUP BY Semester, CourseID
) AS CourseEnrollments
GROUP BY Semester
ORDER BY AvgEnrollments DESC
LIMIT 1;

-- 4. List Students with More Than 3 Enrollments
```

```
SELECT StudentID, COUNT(CourseID) AS TotalEnrollments
FROM Enrollment
GROUP BY StudentID
HAVING COUNT(CourseID) > 3;

-- 5. List All Courses and the Number of Students Enrolled in Each
SELECT c.CourseName, COUNT(e.StudentID) AS TotalStudents
FROM Courses c
LEFT JOIN Enrollment e ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.CourseName;
```

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)

```
-- Create Tables
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(255),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(100),
    Email VARCHAR(255)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(255),
```

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

CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY AUTO_INCREMENT,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    Grade DECIMAL(3,2),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

CREATE TABLE Professors (
    ProfessorID INT PRIMARY KEY,
    Name VARCHAR(255),
    Department VARCHAR(100),
    Email VARCHAR(255)
);

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

-- 2. Identify students who failed more than 2 courses (assuming grade < 2.0 is fail)
SELECT s.StudentID, s.Name, COUNT(e.CourseID) AS FailedCourses
FROM Students s
```

```
JOIN Enrollment e ON s.StudentID = e.StudentID
WHERE e.Grade < 2.0
GROUP BY s.StudentID, s.Name
HAVING COUNT(e.CourseID) > 2;

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

-- 4. Find courses with zero enrollments
SELECT c.CourseName
FROM Courses c
LEFT JOIN Enrollment e ON c.CourseID = e.CourseID
WHERE e.CourseID IS NULL;

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

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

```
-- Create Tables
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(255),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(100),
    Email VARCHAR(255)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(255),
    Credits INT,
    Department VARCHAR(100)
);

CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY AUTO_INCREMENT,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    Grade DECIMAL(3,2),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

CREATE TABLE Professors (
    ProfessorID INT PRIMARY KEY,
    Name VARCHAR(255),
    Department VARCHAR(100),
```

```
    Email VARCHAR(255)
);

-- 1. Find Students Who Have Not Enrolled in Any Course
SELECT s.StudentID, s.Name
FROM Students s
LEFT JOIN Enrollment e ON s.StudentID = e.StudentID
WHERE e.StudentID IS NULL;

-- 2. Find Students Who Are Enrolled in More Than 3 Courses
SELECT s.StudentID, s.Name, COUNT(e.CourseID) AS TotalEnrollments
FROM Students s
JOIN Enrollment e ON s.StudentID = e.StudentID
GROUP BY s.StudentID, s.Name
HAVING COUNT(e.CourseID) > 3;

-- 3. Find the Average Grade of Students Per Course
SELECT c.CourseName, AVG(e.Grade) AS AverageGrade
FROM Courses c
JOIN Enrollment e ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.CourseName;

-- 4. Retrieve the Highest Grade in Each Course
SELECT c.CourseName, MAX(e.Grade) AS HighestGrade
FROM Courses c
JOIN Enrollment e ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.CourseName;

-- 5. Get the Department with the Highest Number of Students
SELECT Department, COUNT(StudentID) AS TotalStudents
FROM Students
```

GROUP BY Department

ORDER BY TotalStudents DESC

LIMIT 1;

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)

```
-- Create Tables
CREATE TABLE Customer (
    customer_id INT PRIMARY KEY,
    name VARCHAR(255),
    address VARCHAR(255),
    phone VARCHAR(20),
    email VARCHAR(255)
);

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 TABLE Branch (
    branch_id INT PRIMARY KEY,
    branch_name VARCHAR(255),
```

```
    location VARCHAR(255),
    manager_id INT
);

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 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 TABLE Employee (
    employee_id INT PRIMARY KEY,
    name VARCHAR(255),
    position VARCHAR(255),
    branch_id INT,
    salary DECIMAL(15,2),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);
```

```
-- 1. List All Customers and Their Account Details
SELECT c.customer_id, c.name, c.address, c.phone, c.email,
    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 TotalBalance
FROM Branch b
JOIN Account a ON b.branch_id = a.branch_id
GROUP BY b.branch_id, 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 *
FROM Transaction
WHERE account_id = 101
ORDER BY transaction_date DESC;

-- 5. Find Customers Who Have Both a Loan and an Account
SELECT c.customer_id, c.name
FROM Customer c
WHERE c.customer_id IN (SELECT customer_id FROM Account)
AND c.customer_id IN (SELECT customer_id FROM Loan);

-- 6. Create a View of High-Value Customers (Balance > 1,00,000)
```

```
CREATE VIEW HighValueCustomers AS
SELECT c.customer_id, c.name, a.balance
FROM Customer c
JOIN Account a ON c.customer_id = a.customer_id
WHERE a.balance > 100000;
```

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

```
-- Create Tables
CREATE TABLE Customer (
    customer_id INT PRIMARY KEY,
    name VARCHAR(255),
    address VARCHAR(255),
    phone VARCHAR(20),
    email VARCHAR(255)
);

CREATE TABLE Account (
    account_id INT PRIMARY KEY,
    customer_id INT,
    balance DECIMAL(15,2),
    branch_id INT,
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

CREATE TABLE Loan (
```

```
    loan_id INT PRIMARY KEY,
    customer_id INT,
    amount DECIMAL(15,2),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

CREATE TABLE Transaction (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    amount DECIMAL(15,2),
    transaction_date DATE,
    FOREIGN KEY (account_id) REFERENCES Account(account_id)
);

-- 1. List All Customers and Their Account Details
SELECT * FROM Customer
JOIN Account ON Customer.customer_id = Account.customer_id;

-- 2. Find the Total Balance in Each Branch
SELECT branch_id, SUM(balance) AS TotalBalance
FROM Account
GROUP BY branch_id;

-- 3. Find Customers Who Have Taken Loans Greater Than Rs. 1,00,000
SELECT customer_id, name, amount
FROM Customer
JOIN Loan ON Customer.customer_id = Loan.customer_id
WHERE amount > 100000;

-- 4. Retrieve Transaction History for a Specific Account (e.g., Account ID: 101)
SELECT * FROM Transaction
```

```
WHERE account_id = 101;

-- 5. Find Customers Who Have Both a Loan and an Account
SELECT DISTINCT Customer.customer_id, name
FROM Customer
JOIN Account ON Customer.customer_id = Account.customer_id
JOIN Loan ON Customer.customer_id = Loan.customer_id;

-- 6. Create a View of High-Value Customers (Balance > 1,00,000)
CREATE VIEW HighValueCustomers AS
SELECT customer_id, name, balance
FROM Customer
JOIN Account ON Customer.customer_id = Account.customer_id
WHERE balance > 100000;
```