1. **Write a PL/SQL block to calculate the grade of minimum 10 students.**

```
CREATE TABLE students (
  student_id INT,
  name VARCHAR(50),
  marks INT
);
```

```
INSERT INTO students VALUES
(1, 'Amit', 80),
(2, 'Sara', 65),
(3, 'Ravi', 45),
(4, 'Neha', 72),
(5, 'John', 55),
(6, 'Kiran', 30),
(7, 'Meena', 90),
(8, 'Tom', 68),
(9, 'Rita', 58),
(10, 'Rahul', 77);
```

```
DELIMITER //

CREATE PROCEDURE SimpleGradeCalculator()
BEGIN
  DECLARE sid INT;
  DECLARE mark INT;
  DECLARE grade CHAR(1);
  DECLARE done INT DEFAULT FALSE;
  DECLARE cur CURSOR FOR SELECT student_id, marks FROM students;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO sid, mark;
    IF done THEN
      LEAVE read_loop;
    END IF;

    IF mark >= 75 THEN
      SET grade = 'A';
    ELSEIF mark >= 60 THEN
      SET grade = 'B';
    ELSEIF mark >= 50 THEN
      SET grade = 'C';
    ELSE
```

```
    SET grade = 'F';
   END IF;

   SELECT CONCAT('Student ID: ', sid, ', Grade: ', grade) AS result;
  END LOOP;
  CLOSE cur;
END//

DELIMITER ;
```

```
CALL SimpleGradeCalculator();
```

---

2. **Write a procedure - to add new employee into emp table - which will return number of employees working in the department. Pass the dept no.**

```
CREATE TABLE emp (
  emp_id INT AUTO_INCREMENT PRIMARY KEY,
  ename VARCHAR(50),
  salary DECIMAL(10,2),
  dept_no INT
);
```

```
DELIMITER //

CREATE PROCEDURE AddEmployeeAndCount(
    IN p_ename VARCHAR(50),
    IN p_salary DECIMAL(10,2),
    IN p_dept_no INT,
    OUT emp_count INT
)
BEGIN
    -- Insert the new employee
    INSERT INTO emp (ename, salary, dept_no)
    VALUES (p_ename, p_salary, p_dept_no);

    -- Count employees in the given department
    SELECT COUNT(*) INTO emp_count
    FROM emp
    WHERE dept_no = p_dept_no;
END //

DELIMITER ;
```

```
SET @count = 0;
CALL AddEmployeeAndCount('Vinita', 40000, 10, @count);
SELECT @count AS DepartmentEmployeeCount;
```

---

3. **Write a function - that accepts employee number and return the salary status as low, high, based on his salary.**

```
CREATE TABLE emp (
  emp_id INT PRIMARY KEY,
  ename VARCHAR(100),
  salary DECIMAL(10,2),
  dept_no INT
);
```

```
-- Insert sample employees
INSERT INTO emp (emp_id, ename, salary, dept_no) VALUES
(101, 'Alice', 25000, 10),
(102, 'Bob', 45000, 20),
(103, 'Charlie', 70000, 30),
(104, 'David', 30000, 20),
(105, 'Eva', 60000, 10);
```

```
DELIMITER //

CREATE FUNCTION GetSalaryStatus(emp_no INT)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
  DECLARE emp_salary INT;
  DECLARE status VARCHAR(20);

  -- Get the employee's salary
  SELECT salary INTO emp_salary
  FROM emp
  WHERE emp_id = emp_no;

  -- Check if salary is found
  IF emp_salary IS NULL THEN
    SET status = 'Not Found';
  ELSEIF emp_salary < 30000 THEN
    SET status = 'Low';
  ELSEIF emp_salary >= 30000 AND emp_salary < 60000 THEN
    SET status = 'Medium';
  ELSE
    SET status = 'High';
  END IF;

  RETURN status;
END //
```

```
DELIMITER ;
```

```
SELECT GetSalaryStatus(101);  -- Low
SELECT GetSalaryStatus(102);  -- Medium
SELECT GetSalaryStatus(103);  -- High
SELECT GetSalaryStatus(999);  -- Not Found
```

---

4. **Write a function - which will return number of employees working in the department. Pass the dept no.**

```
DROP TABLE IF EXISTS emp;

CREATE TABLE emp (
  emp_id INT PRIMARY KEY,
  emp_name VARCHAR(100),
  salary DECIMAL(10, 2),
  dept_id INT
);
```

```
INSERT INTO emp (emp_id, emp_name, salary, dept_id) VALUES
(1, 'John', 45000, 10),
(2, 'Alice', 55000, 20),
(3, 'Bob', 30000, 10),
(4, 'Maya', 60000, 30),
(5, 'Ravi', 70000, 20);
```

```
DELIMITER //

CREATE FUNCTION CountEmployeesInDept(dept_no INT)
RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE emp_count INT;

  -- Check and count employees in the department
  SELECT COUNT(*) INTO emp_count
  FROM emp
  WHERE dept_id = dept_no;

  -- Return count
  RETURN emp_count;
END //

DELIMITER ;
```

```
-- Count employees in department 10
SELECT CountEmployeesInDept(10); -- Output: 2

-- Count employees in a department with no employees (e.g., 40)
SELECT CountEmployeesInDept(40); -- Output: 0
```

5. **Write a function – which will show the level of the customer whether platinum, gold or silver.**

```
DROP TABLE IF EXISTS customers;

CREATE TABLE customers (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(100),
  total_purchase DECIMAL(10,2)
);
```

```
INSERT INTO customers (customer_id, customer_name, total_purchase) VALUES
(1, 'Ananya', 95000),
(2, 'Rahul', 45000),
(3, 'Sneha', 15000),
(4, 'Aman', 2000);
```

```
DELIMITER //

CREATE FUNCTION GetCustomerLevel(cust_id INT)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
  DECLARE purchase_amount DECIMAL(10,2);
  DECLARE level VARCHAR(20);

  -- Fetch total purchase amount
  SELECT total_purchase INTO purchase_amount
  FROM customers
  WHERE customer_id = cust_id;

  -- Decide level based on amount
  IF purchase_amount >= 80000 THEN
    SET level = 'Platinum';
  ELSEIF purchase_amount >= 40000 THEN
    SET level = 'Gold';
  ELSEIF purchase_amount >= 10000 THEN
    SET level = 'Silver';
  ELSE
    SET level = 'No Level';
  END IF;

  RETURN level;
END //
```

```
DELIMITER ;
```

```
-- Check customer level for customer ID 1
SELECT GetCustomerLevel(1);  -- Output: Platinum

-- For customer ID 3
SELECT GetCustomerLevel(3);  -- Output: Silver

-- For customer ID 4
SELECT GetCustomerLevel(4);  -- Output: No Level
```

---

6. **Write a function which**
   - will accept input as a number and print whether it is even or odd
   - will find the largest number among three numbers.

```
DELIMITER //

CREATE FUNCTION check_even_odd(n INT)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
  DECLARE result VARCHAR(20);

  IF n IS NULL THEN
    SET result = 'Invalid Input';
  ELSEIF n = 0 THEN
    SET result = 'Zero';
  ELSEIF MOD(n, 2) = 0 THEN
    SET result = 'Even';
  ELSE
    SET result = 'Odd';
  END IF;

  RETURN result;
END //

DELIMITER ;
```

```
DELIMITER //

CREATE FUNCTION find_largest(n1 INT, n2 INT, n3 INT)
RETURNS VARCHAR(30)
DETERMINISTIC
BEGIN
  DECLARE result VARCHAR(30);

  IF n1 IS NULL OR n2 IS NULL OR n3 IS NULL THEN
    SET result = 'Invalid input';
  ELSEIF n1 = n2 AND n2 = n3 THEN
    SET result = 'All numbers are equal';
  ELSEIF n1 >= n2 AND n1 >= n3 THEN
    SET result = 'Number1 is largest';
  ELSEIF n2 >= n1 AND n2 >= n3 THEN
    SET result = 'Number2 is largest';
  ELSE
    SET result = 'Number3 is largest';
  END IF;
```

```
  RETURN result;
END //

DELIMITER ;
```

```
SELECT find_largest(10, 5, 3);    -- Number1 is largest
SELECT find_largest(3, 8, 8);     -- Number2 is largest
SELECT find_largest(9, 9, 9);     -- All numbers are equal
SELECT find_largest(NULL, 5, 3);  -- Invalid input
```

```
DELIMITER //

CREATE FUNCTION get_largest_number(n1 INT, n2 INT, n3 INT)
RETURNS VARCHAR(50)
DETERMINISTIC
BEGIN
  DECLARE result VARCHAR(50);

  IF n1 IS NULL OR n2 IS NULL OR n3 IS NULL THEN
    SET result = 'Invalid input';
  ELSEIF n1 = n2 AND n2 = n3 THEN
    SET result = 'All numbers are equal';
  ELSE
    SET result = CONCAT('Largest number is ', GREATEST(n1, n2, n3));
  END IF;

  RETURN result;
END //

DELIMITER ;
```

```
SELECT get_largest_number(40, 90, 30);
```

---

7. Execute cursor without handler and with handler.

1. ✅ **Without handler** (basic example)
2. ✅ **With handler** (to safely handle end-of-data condition and avoid errors)

```
CREATE TABLE employees (
   emp_id INT,
   emp_name VARCHAR(50),
   salary DECIMAL(10, 2)
);

-- Sample data
INSERT INTO employees VALUES (1, 'Alice', 50000);
INSERT INTO employees VALUES (2, 'Bob', 60000);
INSERT INTO employees VALUES (3, 'Charlie', 70000);
```

🔹 **1. Cursor WITHOUT Handler (⚠️ Risk of runtime error if not careful)**

```
DELIMITER //

CREATE PROCEDURE cursor_without_handler()
BEGIN
 DECLARE v_id INT;
 DECLARE v_name VARCHAR(50);
 DECLARE v_salary DECIMAL(10,2);

 DECLARE emp_cursor CURSOR FOR
  SELECT emp_id, emp_name, salary FROM employees;

 OPEN emp_cursor;

 FETCH emp_cursor INTO v_id, v_name, v_salary;

 WHILE v_id IS NOT NULL DO
  SELECT CONCAT('ID: ', v_id, ', Name: ', v_name, ', Salary: ', v_salary);
  FETCH emp_cursor INTO v_id, v_name, v_salary;
 END WHILE;

 CLOSE emp_cursor;
END //

DELIMITER ;
```

❌ *Problem:* This may crash if cursor reaches end of data, since there's no condition handler for end-of-data.

```
call cursor_without_handler();
```

🔹 **2. Cursor WITH Handler (✅ Recommended)**

```
DELIMITER //

CREATE PROCEDURE cursor_with_handler()
BEGIN
  DECLARE v_id INT;
  DECLARE v_name VARCHAR(50);
  DECLARE v_salary DECIMAL(10,2);
  DECLARE done INT DEFAULT FALSE;

  -- Handler for end of cursor
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  DECLARE emp_cursor CURSOR FOR
    SELECT emp_id, emp_name, salary FROM employees;

  OPEN emp_cursor;

  read_loop: LOOP
    FETCH emp_cursor INTO v_id, v_name, v_salary;
    IF done THEN
      LEAVE read_loop;
    END IF;
    SELECT CONCAT('ID: ', v_id, ', Name: ', v_name, ', Salary: ', v_salary);
  END LOOP;

  CLOSE emp_cursor;
END //

DELIMITER ;
```

✅ **Usage:**

```
CALL cursor_with_handler();
```

---

8. Write a procedure - To perform arithmetic operations.

```
DELIMITER //

CREATE PROCEDURE arithmetic_operations(
   IN num1 DOUBLE,
   IN num2 DOUBLE
)
BEGIN
 DECLARE result_add DOUBLE;
 DECLARE result_sub DOUBLE;
 DECLARE result_mul DOUBLE;
 DECLARE result_div VARCHAR(100);
 DECLARE result_mod VARCHAR(100);

 -- Check for NULL inputs
 IF num1 IS NULL OR num2 IS NULL THEN
  SELECT 'Invalid input: one or both numbers are NULL' AS Error;
 ELSE
  -- Perform operations
  SET result_add = num1 + num2;
  SET result_sub = num1 - num2;
  SET result_mul = num1 * num2;

  -- Division check
  IF num2 = 0 THEN
   SET result_div = 'Cannot divide by zero';
   SET result_mod = 'Cannot perform modulo by zero';
  ELSE
   SET result_div = CAST(num1 / num2 AS CHAR);
   SET result_mod = CAST(MOD(num1, num2) AS CHAR);
  END IF;

  -- Show results
  SELECT
   result_add AS Addition,
   result_sub AS Subtraction,
   result_mul AS Multiplication,
   result_div AS Division,
   result_mod AS Modulo;
 END IF;
END;
//

DELIMITER ;
```

```
CALL arithmetic_operations(10, 2);
```

```
CALL arithmetic_operations(10, 0); -- to test divide by zero
CALL arithmetic_operations(NULL, 5); -- to test NULL input
```

---

9. **Write a trigger for before insert/ after delete.**

🔸 Sample Tables

```
CREATE TABLE CUSTOMERS (
  ID INT PRIMARY KEY,
  Name VARCHAR(100),
  Age INT
);

CREATE TABLE CUSTOMER_LOG (
  LogID INT AUTO_INCREMENT PRIMARY KEY,
  Action VARCHAR(50),
  CustomerID INT,
  CustomerName VARCHAR(100),
  ActionTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

✅ Trigger 1: BEFORE INSERT

This trigger logs an entry before a customer is inserted.

```
DELIMITER //

CREATE TRIGGER before_customer_insert
BEFORE INSERT ON CUSTOMERS
FOR EACH ROW
BEGIN
  INSERT INTO CUSTOMER_LOG (Action, CustomerID, CustomerName)
  VALUES ('Before Insert', NEW.ID, NEW.Name);
END //

DELIMITER ;
```

---

1. Insert a new customer (this activates the BEFORE INSERT trigger):

```
INSERT INTO CUSTOMERS (ID, Name, Age)
VALUES (1, 'Vinita', 22);
```

2. View the trigger result from the log table:

```
SELECT * FROM CUSTOMER_LOG;
```

✅ Trigger 2: AFTER DELETE

This trigger logs an entry after a customer is deleted.

```
DELIMITER //

CREATE TRIGGER after_customer_delete
AFTER DELETE ON CUSTOMERS
FOR EACH ROW
BEGIN
  INSERT INTO CUSTOMER_LOG (Action, CustomerID, CustomerName)
  VALUES ('After Delete', OLD.ID, OLD.Name);
END //

DELIMITER ;
```

🔍 **To see the results of this trigger:**

1. **Delete a row from CUSTOMERS:**

```
DELETE FROM CUSTOMERS WHERE ID = 1;
```

2. **Then view the contents of CUSTOMER_LOG:**

```
SELECT * FROM CUSTOMER_LOG;
```

🔍 Explanation:

- NEW.column: Refers to the value being inserted.
- OLD.column: Refers to the value being deleted (or updated).
- These triggers automatically create a log entry in CUSTOMER_LOG.

10. For University database execute following queries:

Department (dept_name, building, budget)
Instructor (inst_id, name, salary, dept_name)
Course (course_id, title, credits, dept_name)
Teaches (course_id, inst_id)

• Create a view to find out only instructors who have taught some course.

• Find the names of all instructors whose salary is greater than at least one instructor in biology dept.

• Find the names of all departments whose name includes substring " i "

```
DROP TABLE IF EXISTS Teaches, Course, Instructor, Department;

CREATE TABLE Department (
   dept_name VARCHAR(50) PRIMARY KEY,
   building VARCHAR(50),
   budget DECIMAL(10, 2)
);

CREATE TABLE Instructor (
   inst_id INT PRIMARY KEY,
   name VARCHAR(50),
   salary DECIMAL(10, 2),
   dept_name VARCHAR(50),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Course (
   course_id VARCHAR(10) PRIMARY KEY,
   title VARCHAR(100),
   credits INT,
   dept_name VARCHAR(50),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Teaches (
   course_id VARCHAR(10),
   inst_id INT,
   FOREIGN KEY (course_id) REFERENCES Course(course_id),
   FOREIGN KEY (inst_id) REFERENCES Instructor(inst_id)
);
```

---

1. Insert into Department

```
INSERT INTO Department (dept_name, building, budget) VALUES
('Computer Science', 'Taylor', 90000),
('Biology', 'Watson', 75000),
('Physics', 'Newton', 80000),
('Linguistics', 'Wordsworth', 60000);
```

2. Insert into Instructor

```
INSERT INTO Instructor (inst_id, name, salary, dept_name) VALUES
(101, 'Alice', 90000, 'Computer Science'),
(102, 'Bob', 60000, 'Biology'),
(103, 'Charlie', 80000, 'Physics'),
(104, 'David', 85000, 'Computer Science'),
(105, 'Eva', 70000, 'Biology');
```

3. Insert into Course

```
INSERT INTO Course (course_id, title, credits, dept_name) VALUES
('CS101', 'Intro to CS', 4, 'Computer Science'),
('BIO101', 'Intro to Biology', 3, 'Biology'),
('PHY101', 'Physics Basics', 4, 'Physics');
```

4. Insert into Teaches

```
INSERT INTO Teaches (course_id, inst_id) VALUES
('CS101', 101),
('BIO101', 102),
('PHY101', 103);
```

🔹 **1. Create a view to find instructors who have taught some course**

```
CREATE VIEW Instructors_Taught_Courses AS
SELECT DISTINCT I.inst_id, I.name, I.dept_name
FROM Instructor I
JOIN Teaches T ON I.inst_id = T.inst_id;
```

✅ **Usage:**

```
SELECT * FROM Instructors_Taught_Courses;
```

🔹 **2. Find names of instructors whose salary is greater than at least one instructor in Biology dept**

```
SELECT name
FROM Instructor
WHERE salary > ANY (
  SELECT salary
  FROM Instructor
  WHERE dept_name = 'Biology'
);
```

🔹 **3. Find names of all departments whose name includes substring 'i'**

```
SELECT dept_name
FROM Department
WHERE dept_name LIKE '%i%';
```

---

11. **For University database execute following queries:**
Department (dept_name, building, budget)
Instructor (inst_id, name, salary, dept_name)
Course (course_id, title, credits, dept_name)
Teaches (course_id, inst_id)
- **Find the names of all instructors in Computer dept who have salary greater than 70000.**
- **Create a view using more than two tables**
- **Find the names of instructors whose names are exactly five characters.**

🔸 **Step 1: Create Tables**

```
CREATE TABLE Department (
   dept_name VARCHAR(50) PRIMARY KEY,
   building VARCHAR(50),
   budget DECIMAL(10, 2)
);

CREATE TABLE Instructor (
   inst_id INT PRIMARY KEY,
   name VARCHAR(50),
   salary DECIMAL(10, 2),
   dept_name VARCHAR(50),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Course (
   course_id VARCHAR(10) PRIMARY KEY,
   title VARCHAR(100),
   credits INT,
   dept_name VARCHAR(50),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Teaches (
   course_id VARCHAR(10),
   inst_id INT,
   FOREIGN KEY (course_id) REFERENCES Course(course_id),
   FOREIGN KEY (inst_id) REFERENCES Instructor(inst_id)
);
```

🔸 **Step 2: Insert Sample Data**

```
-- Insert into Department
INSERT INTO Department (dept_name, building, budget) VALUES
('Computer Science', 'Taylor', 120000.00),
('Biology', 'Watson', 95000.00),
('Physics', 'Newton', 110000.00);
```

```
-- Insert into Instructor
INSERT INTO Instructor (inst_id, name, salary, dept_name) VALUES
(101, 'Alice', 90000.00, 'Computer Science'),
(102, 'Bob', 60000.00, 'Biology'),
(103, 'Carol', 80000.00, 'Physics'),
(104, 'David', 85000.00, 'Computer Science'),
(105, 'Evaaa', 70000.00, 'Biology'),
(106, 'Steve', 71000.00, 'Computer Science');

-- Insert into Course
INSERT INTO Course (course_id, title, credits, dept_name) VALUES
('CS101', 'Intro to CS', 4, 'Computer Science'),
('BIO101', 'Intro to Biology', 3, 'Biology'),
('PHY101', 'Physics Basics', 4, 'Physics');

-- Insert into Teaches
INSERT INTO Teaches (course_id, inst_id) VALUES
('CS101', 101),
('BIO101', 102),
('PHY101', 103),
('CS101', 104),
('CS101', 106);
```

🔸 **Step 3: Queries**

✅ **1. Find instructors in Computer department with salary > 70000**

```
SELECT name
FROM Instructor
WHERE dept_name = 'Computer Science' AND salary > 70000;
```

✅ **2. Create a view using more than two tables**

```
CREATE VIEW Instructor_Course_View AS
SELECT i.name AS instructor_name, c.title AS course_title, d.dept_name AS department
FROM Instructor i
JOIN Teaches t ON i.inst_id = t.inst_id
JOIN Course c ON t.course_id = c.course_id
JOIN Department d ON i.dept_name = d.dept_name;
```

To display the view:

```
SELECT * FROM Instructor_Course_View;
```

✅ **3. Find names of instructors with names exactly 5 characters**

```
SELECT name
FROM Instructor
WHERE CHAR_LENGTH(name) = 5;
```

---

12. **For University database execute following queries:**
Department (dept_name, building, budget)
Instructor (inst_id, name, salary, dept_name)
Course (course_id, title, credits, dept_name)
Teaches (course_id, inst_id)
• Create a view to find instructor name and course for instructors in IT department.
• Find the names of all instructors whose salary is greater than at least one instructor in biology dept.
• Find titles along with department where department must end with " y ".
• Find the titles along with department name of biology department.

✅ **1. Create Tables**

If not already created, here are the table creation statements:

```
CREATE TABLE Department (
    dept_name VARCHAR(50) PRIMARY KEY,
    building VARCHAR(50),
    budget DECIMAL(10, 2)
);

CREATE TABLE Instructor (
    inst_id INT PRIMARY KEY,
    name VARCHAR(50),
    salary DECIMAL(10, 2),
    dept_name VARCHAR(50),
    FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Course (
    course_id VARCHAR(10) PRIMARY KEY,
    title VARCHAR(100),
    credits INT,
    dept_name VARCHAR(50),
    FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Teaches (
    course_id VARCHAR(10),
    inst_id INT,
    FOREIGN KEY (course_id) REFERENCES Course(course_id),
    FOREIGN KEY (inst_id) REFERENCES Instructor(inst_id)
);
```

✅ **2. Insert Sample Data**

```
-- Departments (including IT for this query)
INSERT INTO Department (dept_name, building, budget) VALUES
```

```
('Computer Science', 'Taylor', 120000.00),
('Biology', 'Watson', 95000.00),
('Physics', 'Newton', 110000.00),
('IT', 'Smith', 105000.00);

-- Instructors
INSERT INTO Instructor (inst_id, name, salary, dept_name) VALUES
(101, 'Alice', 90000.00, 'Computer Science'),
(102, 'Bob', 60000.00, 'Biology'),
(103, 'Carol', 80000.00, 'Physics'),
(104, 'David', 85000.00, 'IT'),
(105, 'Eva', 75000.00, 'Biology'),
(106, 'John', 95000.00, 'IT');

-- Courses
INSERT INTO Course (course_id, title, credits, dept_name) VALUES
('CS101', 'Intro to CS', 4, 'Computer Science'),
('BIO101', 'Intro to Biology', 3, 'Biology'),
('PHY101', 'Physics Basics', 4, 'Physics'),
('IT101', 'IT Fundamentals', 3, 'IT');

-- Teaches
INSERT INTO Teaches (course_id, inst_id) VALUES
('CS101', 101),
('BIO101', 102),
('PHY101', 103),
('IT101', 104),
('IT101', 106);
```

🔹 **Query 1: View for instructor name and course in IT department**

```
CREATE VIEW IT_Instructor_Courses AS
SELECT i.name AS instructor_name, c.title AS course_title
FROM Instructor i
JOIN Teaches t ON i.inst_id = t.inst_id
JOIN Course c ON c.course_id = t.course_id
WHERE i.dept_name = 'IT';
```

To see the result:

```
SELECT * FROM IT_Instructor_Courses;
```

🔹 **Query 2: Instructors with salary greater than at least one Biology instructor**

```
SELECT name
FROM Instructor
WHERE salary > ANY (
  SELECT salary
  FROM Instructor
  WHERE dept_name = 'Biology'
);
```

- **Query 3: Titles with departments ending in “y”**

```
SELECT title, dept_name
FROM Course
WHERE dept_name LIKE '%y';
```

- **Query 4: Titles and departments for Biology**

```
SELECT title, dept_name
FROM Course
WHERE dept_name = 'Biology';
```

---

13. For University database execute following queries:

Department (dept_name, building, budget)
Instructor (inst_id, name, salary, dept_name)
Course (course_id, title, credits, dept_name)
Teaches (course_id, inst_id)

- **Find the average salary of the instructors who are in music dept.**
- **Find the average salary in each dept.**
- **Find out department name with average salary in each department where average salary is greater than 40000.**

Step 1: Create Tables

```
-- Create Department table
CREATE TABLE Department (
    dept_name VARCHAR(100) PRIMARY KEY,
    building VARCHAR(100),
    budget DECIMAL(10, 2)
);

-- Create Instructor table
CREATE TABLE Instructor (
    inst_id INT PRIMARY KEY,
    name VARCHAR(100),
    salary DECIMAL(10, 2),
    dept_name VARCHAR(100),
    FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

-- Create Course table
CREATE TABLE Course (
    course_id VARCHAR(10) PRIMARY KEY,
    title VARCHAR(100),
    credits INT,
    dept_name VARCHAR(100),
    FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

-- Create Teaches table
CREATE TABLE Teaches (
    course_id VARCHAR(10),
    inst_id INT,
    PRIMARY KEY (course_id, inst_id),
    FOREIGN KEY (course_id) REFERENCES Course(course_id),
    FOREIGN KEY (inst_id) REFERENCES Instructor(inst_id)
);
```

Step 2: Insert Sample Data

```
-- Insert data into Department
```

```
INSERT INTO Department (dept_name, building, budget) VALUES
('Computer Science', 'Building A', 1000000),
('Biology', 'Building B', 500000),
('Physics', 'Building C', 600000),
('Music', 'Building D', 400000);  -- Adding a Music department

-- Insert data into Instructor
INSERT INTO Instructor (inst_id, name, salary, dept_name) VALUES
(101, 'Alice', 90000, 'Computer Science'),
(102, 'Bob', 60000, 'Biology'),
(103, 'Charlie', 80000, 'Physics'),
(104, 'David', 85000, 'Computer Science'),
(105, 'Eva', 70000, 'Biology'),
(106, 'Frank', 45000, 'Music'),  -- Adding a Music instructor
(107, 'Grace', 50000, 'Music');

-- Insert data into Course
INSERT INTO Course (course_id, title, credits, dept_name) VALUES
('CS101', 'Intro to CS', 4, 'Computer Science'),
('BIO101', 'Intro to Biology', 3, 'Biology'),
('PHY101', 'Physics Basics', 4, 'Physics'),
('MUS101', 'Intro to Music', 3, 'Music');  -- Adding a Music course

-- Insert data into Teaches
INSERT INTO Teaches (course_id, inst_id) VALUES
('CS101', 101),
('BIO101', 102),
('PHY101', 103),
('MUS101', 106);  -- Frank teaches Music course
```

Step 3: Execute the Queries

🔹 **1. Find the average salary of instructors who are in the Music department**

```
SELECT AVG(salary) AS avg_salary_music
FROM Instructor
WHERE dept_name = 'Music';
```

🔹 **2. Find the average salary in each department**

```
SELECT dept_name, AVG(salary) AS avg_salary
FROM Instructor
GROUP BY dept_name;
```

🔹 **3. Find department names where average salary > 40000**

```
SELECT dept_name, AVG(salary) AS avg_salary
FROM Instructor
GROUP BY dept_name
HAVING AVG(salary) > 40000;
```

14. **For University database execute following queries:**
    Department (dept_name, building, budget)

Instructor (inst_id, name, salary, dept_name)
Course (course_id, title, credits, dept_name)
Teaches (course_id, inst_id)

- **Find the names of all instructors in music dept who have salary greater than 50000.**
- **Find the details of instructors who are teaching some courses**
- **List all instructors along with the courses that they teach.**
- **List instructors in descending order.**

Step 1: Create Tables

```
-- Create Department table
CREATE TABLE Department (
   dept_name VARCHAR(100) PRIMARY KEY,
   building VARCHAR(100),
   budget DECIMAL(10, 2)
);

-- Create Instructor table
CREATE TABLE Instructor (
   inst_id INT PRIMARY KEY,
   name VARCHAR(100),
   salary DECIMAL(10, 2),
   dept_name VARCHAR(100),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

-- Create Course table
CREATE TABLE Course (
   course_id VARCHAR(10) PRIMARY KEY,
   title VARCHAR(100),
   credits INT,
   dept_name VARCHAR(100),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

-- Create Teaches table
CREATE TABLE Teaches (
   course_id VARCHAR(10),
   inst_id INT,
   PRIMARY KEY (course_id, inst_id),
   FOREIGN KEY (course_id) REFERENCES Course(course_id),
   FOREIGN KEY (inst_id) REFERENCES Instructor(inst_id)
);
```

Step 2: Insert Sample Data

```
-- Insert data into Department
INSERT INTO Department (dept_name, building, budget) VALUES
('Computer Science', 'Building A', 1000000),
```

```
('Biology', 'Building B', 500000),
('Physics', 'Building C', 600000),
('Music', 'Building D', 400000);

-- Insert data into Instructor
INSERT INTO Instructor (inst_id, name, salary, dept_name) VALUES
(101, 'Alice', 90000, 'Computer Science'),
(102, 'Bob', 60000, 'Biology'),
(103, 'Charlie', 80000, 'Physics'),
(104, 'David', 85000, 'Computer Science'),
(105, 'Eva', 70000, 'Biology'),
(106, 'Frank', 45000, 'Music'),
(107, 'Grace', 55000, 'Music');

-- Insert data into Course
INSERT INTO Course (course_id, title, credits, dept_name) VALUES
('CS101', 'Intro to CS', 4, 'Computer Science'),
('BIO101', 'Intro to Biology', 3, 'Biology'),
('PHY101', 'Physics Basics', 4, 'Physics'),
('MUS101', 'Intro to Music', 3, 'Music');

-- Insert data into Teaches
INSERT INTO Teaches (course_id, inst_id) VALUES
('CS101', 101),
('BIO101', 102),
('PHY101', 103),
('MUS101', 106); -- Frank teaches Music course
```

Step 3: Execute the Queries

Now that the tables are created and data is inserted, you can execute the following queries:

🔹 **1. Find the names of all instructors in the Music department who have a salary greater than 50,000**

```
SELECT name
FROM Instructor
WHERE dept_name = 'Music' AND salary > 50000;
```

🔹 **2. Find the details of instructors who are teaching some courses**

```
SELECT I.*
FROM Instructor I
JOIN Teaches T ON I.inst_id = T.inst_id;
```

🔹 **3. List all instructors along with the courses that they teach**

```
SELECT I.name, C.title
FROM Instructor I
JOIN Teaches T ON I.inst_id = T.inst_id
JOIN Course C ON T.course_id = C.course_id;
```

- **4. List instructors in descending order of their salary**

```
SELECT name, salary
FROM Instructor
ORDER BY salary DESC;
```

15. For University database execute following queries:

Department (dept_name, building, budget)
Instructor (inst_id, name, salary, dept_name)
Course (course_id, title, credits, dept_name)
Teaches (course_id, inst_id)

- **Find the names of instructors who are working in IT dept.**
- **Create a view to find out only instructors who have taught some course.**
- **Give the increment of rs. 10000 to instructors whose salary is less than 40000 else give increment of rs.20000.**
- **Find average salary in each department.**

Step 1: Create the Tables

```
-- Create Department table
CREATE TABLE Department (
    dept_name VARCHAR(100) PRIMARY KEY,
    building VARCHAR(100),
    budget DECIMAL(10, 2)
);

-- Create Instructor table
CREATE TABLE Instructor (
    inst_id INT PRIMARY KEY,
    name VARCHAR(100),
    salary DECIMAL(10, 2),
    dept_name VARCHAR(100),
    FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

-- Create Course table
CREATE TABLE Course (
    course_id VARCHAR(10) PRIMARY KEY,
    title VARCHAR(100),
    credits INT,
    dept_name VARCHAR(100),
    FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

-- Create Teaches table
CREATE TABLE Teaches (
    course_id VARCHAR(10),
    inst_id INT,
    PRIMARY KEY (course_id, inst_id),
    FOREIGN KEY (course_id) REFERENCES Course(course_id),
    FOREIGN KEY (inst_id) REFERENCES Instructor(inst_id)
);
```

Step 2: Insert Sample Data

```
-- Insert data into Department
INSERT INTO Department (dept_name, building, budget) VALUES
('Computer Science', 'Building A', 1000000),
('Biology', 'Building B', 500000),
('Physics', 'Building C', 600000),
('IT', 'Building D', 400000);  -- IT department

-- Insert data into Instructor
INSERT INTO Instructor (inst_id, name, salary, dept_name) VALUES
(101, 'Alice', 90000, 'Computer Science'),
(102, 'Bob', 60000, 'Biology'),
(103, 'Charlie', 80000, 'Physics'),
(104, 'David', 35000, 'IT'),
(105, 'Eva', 70000, 'Biology'),
(106, 'Frank', 45000, 'IT'),
(107, 'Grace', 55000, 'IT');

-- Insert data into Course
INSERT INTO Course (course_id, title, credits, dept_name) VALUES
('CS101', 'Intro to CS', 4, 'Computer Science'),
('BIO101', 'Intro to Biology', 3, 'Biology'),
('PHY101', 'Physics Basics', 4, 'Physics'),
('IT101', 'Intro to IT', 3, 'IT');

-- Insert data into Teaches
INSERT INTO Teaches (course_id, inst_id) VALUES
('CS101', 101),
('BIO101', 102),
('PHY101', 103),
('IT101', 106);  -- Frank teaches IT101 course
```

Step 3: Execute the Queries

- **1. Find the names of instructors who are working in the IT department**

```
SELECT name
FROM Instructor
WHERE dept_name = 'IT';
```

- **2. Create a view to find out only instructors who have taught some course**

```
CREATE VIEW InstructorsWhoTaught AS
SELECT DISTINCT I.name
FROM Instructor I
JOIN Teaches T ON I.inst_id = T.inst_id;
```

```
SELECT * FROM InstructorsWhoTaught;
```

- **3. Give the increment of Rs. 10,000 to instructors whose salary is less than Rs. 40,000, else give an increment of Rs. 20,000**

```
UPDATE Instructor
SET salary = CASE
   WHEN salary < 40000 THEN salary + 10000
   ELSE salary + 20000
END;
```

- **4. Find the average salary in each department**

```
SELECT dept_name, AVG(salary) AS avg_salary
FROM Instructor
GROUP BY dept_name;
```

---

16. **For University database execute following queries:**
Department (dept_name, building, budget)
Instructor (inst_id, name, salary, dept_name)
Course (course_id, title, credits, dept_name) Teaches (course_id, inst_id)
- **Find average salary in each department**
- **Find number of instructors with department name in each department**
- **Find the names of all departments whose name includes substring " i ".**
- **List the entire instructor relation in descending order.**

✅ **1. Create Tables**

```
CREATE TABLE Department (
   dept_name VARCHAR(50) PRIMARY KEY,
   building VARCHAR(50),
   budget DECIMAL(10, 2)
);

CREATE TABLE Instructor (
   inst_id INT PRIMARY KEY,
   name VARCHAR(50),
   salary DECIMAL(10, 2),
   dept_name VARCHAR(50),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Course (
   course_id VARCHAR(10) PRIMARY KEY,
   title VARCHAR(100),
   credits INT,
   dept_name VARCHAR(50),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Teaches (
   course_id VARCHAR(10),
   inst_id INT,
   PRIMARY KEY (course_id, inst_id),
   FOREIGN KEY (course_id) REFERENCES Course(course_id),
   FOREIGN KEY (inst_id) REFERENCES Instructor(inst_id)
);
```

✅ **2. Insert Sample Data**

```
-- Department
INSERT INTO Department VALUES
('Computer Science', 'Block A', 100000),
('Biology', 'Block B', 80000),
('Physics', 'Block C', 90000),
('Music', 'Block D', 75000),
```

('IT', 'Block E', 95000);

-- Instructor
INSERT INTO Instructor VALUES
(1, 'Alice', 55000, 'Computer Science'),
(2, 'Bob', 30000, 'Biology'),
(3, 'Charlie', 70000, 'Physics'),
(4, 'David', 40000, 'Music'),
(5, 'Eva', 50000, 'IT'),
(6, 'Nikki', 65000, 'Biology');

-- Course
INSERT INTO Course VALUES
('CS101', 'Intro to CS', 4, 'Computer Science'),
('BIO101', 'Intro to Biology', 3, 'Biology'),
('PHY101', 'Intro to Physics', 4, 'Physics'),
('MUS101', 'Music Theory', 3, 'Music'),
('IT101', 'Networks', 3, 'IT');

-- Teaches
INSERT INTO Teaches VALUES
('CS101', 1),
('BIO101', 2),
('PHY101', 3),
('MUS101', 4),
('IT101', 5);

✅ **3. Queries**

🔹 **a. Find average salary in each department**

```
SELECT dept_name, AVG(salary) AS avg_salary
FROM Instructor
GROUP BY dept_name;
```

🔹 **b. Find number of instructors with department name in each department**

```
SELECT dept_name, COUNT(*) AS num_instructors
FROM Instructor
GROUP BY dept_name;
```

🔹 **c. Find the names of all departments whose name includes substring “i”**

```
SELECT dept_name
FROM Department
WHERE dept_name LIKE '%i%';
```

🔹 **d. List the entire instructor relation in descending order by salary**

```
SELECT * FROM Instructor
ORDER BY salary DESC;
```

17. **Department (dept_name, building, budget)**
Student(sid, sname, tot_credits,dept_name,grade)
Course (course_id, title, credits, dept_name)
Takes(sid,course_id)
- **Find the name of students who have taken some courses**
- **Find the details of the students who are in Computer department**
- **Find the names of all departments whose name includes substring " a ".**

✅ **1. Create Tables**

```
CREATE TABLE Department (
    dept_name VARCHAR(50) PRIMARY KEY,
    building VARCHAR(50),
    budget DECIMAL(10, 2)
);

CREATE TABLE Student (
    sid INT PRIMARY KEY,
    sname VARCHAR(50),
    tot_credits INT,
    dept_name VARCHAR(50),
    grade VARCHAR(2),
    FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Course (
    course_id VARCHAR(10) PRIMARY KEY,
    title VARCHAR(100),
    credits INT,
    dept_name VARCHAR(50),
    FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Takes (
    sid INT,
    course_id VARCHAR(10),
    PRIMARY KEY (sid, course_id),
    FOREIGN KEY (sid) REFERENCES Student(sid),
    FOREIGN KEY (course_id) REFERENCES Course(course_id)
);
```

✅ **2. Insert Sample Data**

```
-- Department
INSERT INTO Department VALUES
('Computer', 'A-Block', 100000),
('Mechanical', 'B-Block', 80000),
('Electrical', 'C-Block', 90000),
('Civil', 'D-Block', 70000);
```

```
-- Student
INSERT INTO Student VALUES
(1, 'Anuj', 120, 'Computer', 'A'),
(2, 'Bhavna', 110, 'Mechanical', 'B'),
(3, 'Karan', 90, 'Computer', 'A'),
(4, 'Sana', 100, 'Civil', 'C');

-- Course
INSERT INTO Course VALUES
('CS101', 'Data Structures', 4, 'Computer'),
('ME101', 'Thermodynamics', 3, 'Mechanical'),
('CE101', 'Construction', 3, 'Civil');

-- Takes
INSERT INTO Takes VALUES
(1, 'CS101'),
(2, 'ME101'),
(4, 'CE101');
```

✅ **3. Queries**

🔹 **a. Find the name of students who have taken some courses**

```
SELECT DISTINCT s.sname
FROM Student s
JOIN Takes t ON s.sid = t.sid;
```

🔹 **b. Find the details of the students who are in Computer department**

```
SELECT *
FROM Student
WHERE dept_name = 'Computer';
```

🔹 **c. Find the names of all departments whose name includes substring “a”**

```
SELECT dept_name
FROM Department
WHERE dept_name LIKE '%a%';
```

---

18. **Consider following database:**
Student (RollNo, Name, Address)
Subject (SubCode, SubName)
Marks (RollNo, SubCode,Marks) Write following queries in SQL: i) Find average marks of each student, along with the name of student. ii) Find how many students have failed in the subject "DBMS". iii) Find the number of students who are passed in " OS". iv) Find the maximum marks of the subject "TOC".

✅ **1. Create Tables**

```
CREATE TABLE Student (
   RollNo INT PRIMARY KEY,
   Name VARCHAR(50),
   Address VARCHAR(100)
);

CREATE TABLE Subject (
   SubCode VARCHAR(10) PRIMARY KEY,
   SubName VARCHAR(50)
);

CREATE TABLE Marks (
   RollNo INT,
   SubCode VARCHAR(10),
   Marks INT,
   FOREIGN KEY (RollNo) REFERENCES Student(RollNo),
   FOREIGN KEY (SubCode) REFERENCES Subject(SubCode)
);
```

✅ **2. Insert Sample Data**

```
-- Students
INSERT INTO Student VALUES
(1, 'Aarav', 'Pune'),
(2, 'Sneha', 'Mumbai'),
(3, 'Riya', 'Delhi'),
(4, 'Kunal', 'Nagpur');

-- Subjects
INSERT INTO Subject VALUES
('S1', 'DBMS'),
('S2', 'OS'),
('S3', 'TOC');

-- Marks
INSERT INTO Marks VALUES
(1, 'S1', 45),
(1, 'S2', 67),
(1, 'S3', 76),
(2, 'S1', 25),  -- Failed in DBMS
```

(2, 'S2', 35),
(2, 'S3', 66),
(3, 'S1', 85),
(3, 'S2', 40),
(3, 'S3', 59),
(4, 'S1', 30), -- Edge case
(4, 'S2', 20), -- Failed in OS
(4, 'S3', 90);
Let's assume **passing marks = 30**

✅ **3. Queries**

🔹 **i) Find average marks of each student, along with the name of the student**

```
SELECT s.Name, AVG(m.Marks) AS Avg_Marks
FROM Student s
JOIN Marks m ON s.RollNo = m.RollNo
GROUP BY s.Name;
```

🔹 **ii) Find how many students have failed in the subject “DBMS”**

```
SELECT COUNT(DISTINCT m.RollNo) AS Failed_Students
FROM Marks m
JOIN Subject s ON m.SubCode = s.SubCode
WHERE s.SubName = 'DBMS' AND m.Marks < 30;
```

🔹 **iii) Find the number of students who are passed in “OS”**

```
SELECT COUNT(DISTINCT m.RollNo) AS Passed_Students
FROM Marks m
JOIN Subject s ON m.SubCode = s.SubCode
WHERE s.SubName = 'OS' AND m.Marks >= 30;
```

🔹 **iv) Find the maximum marks of the subject “TOC”**

```
SELECT MAX(m.Marks) AS Max_Marks_TOC
FROM Marks m
JOIN Subject s ON m.SubCode = s.SubCode
WHERE s.SubName = 'TOC';
```

---

19. Consider a relational database Supplier (Sid, Sname, address) Parts(Pid, Pname, color) Catalog(Sid, Pid, cost)
Write SQL queries for the following:

i) Find the names of suppliers who supply some red parts.
ii) Find the names of all parts whose cost is more than Rs.250.
iii) Find name of all parts whose color is green.
iv) Find number of parts supplied by each supplier

✅ **1. Table Creation**

```
CREATE TABLE Supplier (
   Sid INT PRIMARY KEY,
   Sname VARCHAR(50),
   address VARCHAR(100)
);

CREATE TABLE Parts (
   Pid INT PRIMARY KEY,
   Pname VARCHAR(50),
   color VARCHAR(20)
);

CREATE TABLE Catalog (
   Sid INT,
   Pid INT,
   cost DECIMAL(10, 2),
   FOREIGN KEY (Sid) REFERENCES Supplier(Sid),
   FOREIGN KEY (Pid) REFERENCES Parts(Pid)
);
```

✅ **2. Sample Data**

```
-- Suppliers
INSERT INTO Supplier VALUES
(1, 'Aarav Suppliers', 'Pune'),
```

```
(2, 'Global Traders', 'Mumbai'),
(3, 'EcoParts', 'Delhi');

-- Parts
INSERT INTO Parts VALUES
(101, 'Nut', 'Red'),
(102, 'Bolt', 'Green'),
(103, 'Screw', 'Blue'),
(104, 'Washer', 'Red');

-- Catalog
INSERT INTO Catalog VALUES
(1, 101, 150.00),
(1, 102, 200.00),
(2, 103, 300.00),
(3, 104, 275.00);
```

✅ **3. Queries**

🔹 **i) Find the names of suppliers who supply some red parts**

```
SELECT DISTINCT s.Sname
FROM Supplier s
JOIN Catalog c ON s.Sid = c.Sid
JOIN Parts p ON c.Pid = p.Pid
WHERE p.color = 'Red';
```

🔹 **ii) Find the names of all parts whose cost is more than Rs.250**

```
SELECT DISTINCT p.Pname
FROM Parts p
JOIN Catalog c ON p.Pid = c.Pid
WHERE c.cost > 250;
```

- **iii) Find name of all parts whose color is green**

```
SELECT Pname
FROM Parts
WHERE color = 'Green';
```

- **iv) Find number of parts supplied by each supplier**

```
SELECT s.Sname, COUNT(c.Pid) AS Number_of_Parts
FROM Supplier s
JOIN Catalog c ON s.Sid = c.Sid
GROUP BY s.Sname;
```

---

20. Department (dept_name, building, budget)
Student(sid, sname, tot_credits,dept_name,grade)
Course (course_id, title, credits, dept_name)
Takes(sid,course_id)

- **Find the name of students who have taken some courses**
- **Find the details of the students who are in Computer department**
- **Find the names of all departments whose name includes substring " a ".**

🔧 **Table Creation**

```
CREATE TABLE Department (
   dept_name VARCHAR(50) PRIMARY KEY,
   building VARCHAR(50),
   budget DECIMAL(10, 2)
);

CREATE TABLE Student (
   sid INT PRIMARY KEY,
   sname VARCHAR(50),
   tot_credits INT,
   dept_name VARCHAR(50),
   grade VARCHAR(2),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Course (
   course_id VARCHAR(10) PRIMARY KEY,
   title VARCHAR(100),
   credits INT,
   dept_name VARCHAR(50),
   FOREIGN KEY (dept_name) REFERENCES Department(dept_name)
);

CREATE TABLE Takes (
   sid INT,
   course_id VARCHAR(10),
   FOREIGN KEY (sid) REFERENCES Student(sid),
   FOREIGN KEY (course_id) REFERENCES Course(course_id)
);
```

📥 **Sample Data**

```
-- Departments
INSERT INTO Department VALUES
('Computer', 'Block A', 500000),
('Mathematics', 'Block B', 300000),
('Physics', 'Block C', 400000);

-- Students
```

```
INSERT INTO Student VALUES
(1, 'Alice', 90, 'Computer', 'A'),
(2, 'Bob', 80, 'Mathematics', 'B'),
(3, 'Charlie', 70, 'Physics', 'C'),
(4, 'David', 85, 'Computer', 'A');

-- Courses
INSERT INTO Course VALUES
('CS101', 'Intro to CS', 4, 'Computer'),
('MA101', 'Calculus', 3, 'Mathematics'),
('PH101', 'Mechanics', 3, 'Physics');

-- Takes
INSERT INTO Takes VALUES
(1, 'CS101'),
(2, 'MA101'),
(3, 'PH101'),
(4, 'CS101');
```

✅ **Required Queries**

1. Find the name of students who have taken some courses

```
SELECT DISTINCT s.sname
FROM Student s
JOIN Takes t ON s.sid = t.sid;
```

---

2. Find the details of the students who are in Computer department

```
SELECT *
FROM Student
WHERE dept_name = 'Computer';
```

---

3. Find the names of all departments whose name includes substring “a”

```
SELECT dept_name
FROM Department
WHERE dept_name LIKE '%a%';
```

---

21. Consider a relational database
Supplier (Sid, Sname, address)
Parts(Pid, Pname, color)
Catalog(Sid, Pid, cost)
Write SQL queries for the following:

v) Find the names of suppliers who supply some red parts.
vi) Find the names of all parts whose cost is more than Rs.250.
vii) Find name of all parts whose color is green.
viii) Find number of parts supplied by each supplier

🔧 **Table Creation**

```
CREATE TABLE Supplier (
   Sid INT PRIMARY KEY,
   Sname VARCHAR(50),
   Address VARCHAR(100)
);

CREATE TABLE Parts (
   Pid INT PRIMARY KEY,
   Pname VARCHAR(50),
   Color VARCHAR(20)
);

CREATE TABLE Catalog (
   Sid INT,
   Pid INT,
   Cost DECIMAL(10, 2),
   FOREIGN KEY (Sid) REFERENCES Supplier(Sid),
   FOREIGN KEY (Pid) REFERENCES Parts(Pid)
);
```

📥 **Sample Data**

```
-- Suppliers
INSERT INTO Supplier VALUES
```

```
(1, 'ABC Supplies', 'Pune'),
(2, 'XYZ Traders', 'Mumbai'),
(3, 'LMN Corp', 'Delhi');

-- Parts
INSERT INTO Parts VALUES
(101, 'Bolt', 'Red'),
(102, 'Nut', 'Green'),
(103, 'Washer', 'Blue'),
(104, 'Screw', 'Red');

-- Catalog (Supplier-Parts-Cost)
INSERT INTO Catalog VALUES
(1, 101, 150.00),
(2, 102, 300.00),
(1, 104, 200.00),
(3, 103, 275.00);
```

✅ **Required Queries**

v) Find the names of suppliers who supply some red parts

```
SELECT DISTINCT S.Sname
FROM Supplier S
JOIN Catalog C ON S.Sid = C.Sid
JOIN Parts P ON C.Pid = P.Pid
WHERE P.Color = 'Red';
```

---

vi) Find the names of all parts whose cost is more than Rs. 250

```
SELECT DISTINCT P.Pname
FROM Parts P
JOIN Catalog C ON P.Pid = C.Pid
WHERE C.Cost > 250;
```

---

vii) Find the names of all parts whose color is green

```
SELECT Pname
FROM Parts
WHERE Color = 'Green';
```

---

viii) Find the number of parts supplied by each supplier

```
SELECT S.Sname, COUNT(C.Pid) AS NumberOfParts
FROM Supplier S
JOIN Catalog C ON S.Sid = C.Sid
GROUP BY S.Sname;
```

Stored Procedure Problem Statements:

1. Create Table CUSTOMERS. Insert records in CUSTOMERS Table.
Write a procedure named 'GetCustomerInfo' without any parameters to retrieve all the records from CUSTOMERS table where age is greater than 25.

2. Create Table CUSTOMERS. Insert records in CUSTOMERS Table.
Write a procedure named 'GetCustomerInfo' that takes a customer's ID as an input parameter and returns that customer's details.

3. Create Table CUSTOMERS. Insert records in CUSTOMERS Table.
Write a procedure named 'GetCustomerInfo' that takes customer's ID as an input parameter and returns that customer's SALARY using an output parameter "Cust_Salary".

4. reate Table CUSTOMERS. Insert records in CUSTOMERS Table.
Write a procedure named 'GetCustomerInfo' that retrieves the current salary of the customer using the IN parameter cust_id. It then increases the salary by 10% and stores the increased salary in the INOUT parameter salary

---

✅ **Step 1: Create Table & Insert Records**

```
CREATE TABLE CUSTOMERS (
 ID INT PRIMARY KEY,
 NAME VARCHAR(50),
 AGE INT,
 ADDRESS VARCHAR(100),
 SALARY DECIMAL(10,2)
);


-- Insert sample records
INSERT INTO CUSTOMERS VALUES
(1, 'Amit', 30, 'Pune', 50000),
(2, 'Sara', 22, 'Mumbai', 45000),
(3, 'Ravi', 35, 'Delhi', 60000),
(4, 'Nita', 28, 'Bangalore', 52000);
```

1️⃣ **Procedure without parameters – returns customers with age > 25**

```
DELIMITER //
```

```
CREATE PROCEDURE GetCustomerInfo()
BEGIN
  SELECT * FROM CUSTOMERS
  WHERE AGE > 25;
END //

DELIMITER ;

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

2️⃣ Procedure with IN parameter – return full details of one customer

```
DELIMITER //

CREATE PROCEDURE GetCustomerInfo(IN cust_id INT)
BEGIN
  SELECT * FROM CUSTOMERS
  WHERE ID = cust_id;
END //

DELIMITER ;

-- Example call
CALL GetCustomerInfo(1);
```

3️⃣ Procedure with IN and OUT – return only salary of customer

```
DELIMITER //

CREATE PROCEDURE GetCustomerInfo(
  IN cust_id INT,
```

```
  OUT Cust_Salary DECIMAL(10,2)
)
BEGIN
  SELECT SALARY INTO Cust_Salary
  FROM CUSTOMERS
  WHERE ID = cust_id;
END //

DELIMITER ;

-- Example call
SET @salary = 0;
CALL GetCustomerInfo(1, @salary);
SELECT @salary AS Salary;
```

4️⃣ Procedure with IN and INOUT – increase salary by 10%

```
DELIMITER //

CREATE PROCEDURE GetCustomerInfo(
  IN cust_id INT,
  INOUT salary DECIMAL(10,2)
)
BEGIN
  SELECT SALARY INTO salary
  FROM CUSTOMERS
  WHERE ID = cust_id;

  SET salary = salary * 1.10;
END //

DELIMITER ;
```

```
-- Example call
SET @newsalary = 0;
CALL GetCustomerInfo(1, @newsalary);
SELECT @newsalary AS IncreasedSalary;
```
