**Sree Ganesh M**

**D&A**

**ACE12407**

```
-- database
 CREATE DATABASE Practice;
 USE Practice;



-- Create the `employees` table
 CREATE TABLE employees (
    id INT PRIMARY KEY,              -- Unique identifier for each employee
    name VARCHAR(50),                 -- Employee's name
    salary DECIMAL(10, 2),           -- Employee's salary
    joining_date DATE,              -- Date the employee joined
    department_id INT,               -- Reference to department
    manager_id INT                  -- ID of the manager (self-referencing)
 );

 -- Insert sample data into the `employees` table
 INSERT INTO employees (id, name, salary, joining_date, department_id, manager_id) VALUES
 (1, 'Alice', 60000, '2020-05-15', 101, NULL),
 (2, 'Bob', 45000, '2019-03-20', 102, 1),
 (3, 'Charlie', 75000, '2021-11-10', 103, 1),
 (4, 'Andrew', 50000, '2022-02-01', 104, 2),
 (5, 'Alex', 52000, '2020-07-18', 101, 1);

 -- Create the `departments` table
 CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50)
 );



DROP TABLE departments;

 -- Insert sample data into `departments` table
```

```sql
INSERT INTO departments (department_id, department_name) VALUES
(101, 'HR'),
(102, 'Finance'),
(103, 'IT'),
(104, 'Marketing');

-- Create the `projects` table
CREATE TABLE projects (
    project_id INT PRIMARY KEY,
    project_name VARCHAR(50)
);

-- Insert sample data into `projects` table
INSERT INTO projects (project_id, project_name) VALUES
(201, 'Project A'), (202, 'Project B');

-- Create the `employees_projects` table
CREATE TABLE employees_projects (
    employee_id INT,
    project_id INT
);

-- Insert sample data into `employees_projects` table
INSERT INTO employees_projects (employee_id, project_id) VALUES
(1, 201), (1, 202), (2, 201);


-- 1. Display all rows and columns
SELECT * FROM employees;

-- 2. Retrieve only the name and salary
SELECT name, salary FROM employees;

-- 3. Find all employees whose salary is greater than 50,000
SELECT * FROM employees WHERE salary > 50000;

-- 4. List all employees who joined the company in 2020
SELECT * FROM employees WHERE YEAR(joining_date) = 2020;
```

```
 -- 5. Retrieve employees whose names start with 'A'
 SELECT * FROM employees WHERE name LIKE 'A%';


-- 6. Calculate the average salary
 SELECT AVG(salary) AS average_salary FROM employees;

 -- 7. Total number of employees
 SELECT COUNT(*) AS total_employees FROM employees;

 -- 8. Highest salary in the employees table
 SELECT MAX(salary) AS highest_salary FROM employees;

 -- 9. Total salary paid by the company
 SELECT SUM(salary) AS total_salary FROM employees;

 -- 10. Count of employees in each department
 SELECT department_id, COUNT(*) AS employee_count FROM employees GROUP BY department_id;


-- 11. Employee names with their department names
 SELECT e.name, d.department_name
 FROM employees AS e
 JOIN departments AS d ON e.department_id = d.department_id;

 -- 12. Employees with their managers
 SELECT e.name AS employee, m.name AS manager
 FROM employees AS e
 JOIN employees AS m ON e.manager_id = m.id;

 -- 13. Employees working on multiple projects
 SELECT e.name
 FROM employees AS e
 JOIN employees_projects AS ep ON e.id = ep.employee_id
 GROUP BY e.name
 HAVING COUNT(ep.project_id) > 1;

 -- 14. Projects and the employees assigned to them
 SELECT p.project_name, e.name
```

```sql
FROM projects AS p
JOIN employees_projects AS ep ON p.project_id = ep.project_id
JOIN employees AS e ON ep.employee_id = e.id;

-- 15. Employees who do not belong to any department
SELECT name FROM employees WHERE department_id IS NULL;



-- 16. Employees with the second-highest salary
SELECT name, salary
FROM employees
WHERE salary = (
    SELECT MAX(salary)
    FROM employees
    WHERE salary < (SELECT MAX(salary) FROM employees)
);

-- 17. Employees whose salary is above the department average
SELECT name, salary
FROM employees AS e
WHERE salary > (
    SELECT AVG(salary)
    FROM employees AS sub_e
    WHERE sub_e.department_id = e.department_id
);

-- 18. Employees earning more than the company average
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);

-- 19. Department with the highest number of employees
SELECT department_id, COUNT(*) AS employee_count
FROM employees
GROUP BY department_id
ORDER BY employee_count DESC
LIMIT 1;

-- 20. Employees in departments located in 'New York' (Example: Assuming location column)
```

```
SELECT e.name
FROM employees AS e
JOIN departments AS d ON e.department_id = d.department_id
WHERE d.location = 'New York';


-- 21. Employees in either 'HR' or 'Finance' department
SELECT name
FROM employees
WHERE department_id IN (
    SELECT department_id FROM departments WHERE department_name IN ('HR', 'Finance')
);

-- 22. Employees working on both Project A and Project B
SELECT name
FROM employees AS e
WHERE EXISTS (
SELECT 1 FROM employees_projects AS ep WHERE ep.employee_id = e.id AND ep.project_id = 201
)
AND EXISTS (
SELECT 1 FROM employees_projects AS ep WHERE ep.employee_id = e.id AND ep.project_id = 202
);

-- 23. Employees not assigned to any project
SELECT name
FROM employees
WHERE id NOT IN (
    SELECT employee_id FROM employees_projects
);

-- 24. Unique job titles across all departments
SELECT DISTINCT department_name FROM employees;

-- 25. Combine employees and former_employees without duplicates
SELECT name FROM employees
UNION
SELECT name FROM former_employees;
```

```sql
-- 26. Add a new employee
 INSERT INTO employees (id, name, salary, joining_date, department_id, manager_id)
 VALUES (6, 'Diana', 48000, '2023-03-01', 103, 2);

 -- 27. Update the salary of all employees in 'IT' by 10%
 UPDATE employees
 SET salary = salary * 1.1
 WHERE department_id = (SELECT department_id FROM departments WHERE department_name = 'IT');

 -- 28. Delete employees who haven't worked for more than 5 years
 DELETE FROM employees WHERE DATEDIFF(CURDATE(), joining_date) > 365 * 5;

 -- 29. Create a backup of the `departments` table
 CREATE TABLE departments_backup AS SELECT * FROM departments;

 -- 30. Drop the temporary_data table
 DROP TABLE IF EXISTS temporary_data;



-- 31. Add a primary key to the `employees` table (if not already defined)
 ALTER TABLE employees ADD CONSTRAINT pk_employees PRIMARY KEY (id);

 -- 32. Create a foreign key between `employees` and `departments` tables
 ALTER TABLE employees ADD CONSTRAINT fk_department
 FOREIGN KEY (department_id) REFERENCES departments(department_id);

 -- 33. Add a unique constraint to the email column in the `employees` table
 ALTER TABLE employees ADD COLUMN email VARCHAR(100);
 ALTER TABLE employees ADD CONSTRAINT unique_email UNIQUE (email);

 -- 34. Check all constraints applied on the `employees` table
 SELECT table_name, constraint_name, constraint_type
 FROM information_schema.table_constraints
 WHERE table_name = 'employees';
 -- 35. Remove the NOT NULL constraint from the `phone_number` column in `employees`
 ALTER TABLE employees MODIFY COLUMN phone_number VARCHAR(15) NULL;


-- PL\SQL
 -- 36. PL/SQL: Calculate the factorial of a given number
 DELIMITER $$
 CREATE PROCEDURE CalculateFactorial(IN num INT, OUT result BIGINT)
 BEGIN
     DECLARE i INT DEFAULT 1;
     SET result = 1;
     WHILE i <= num DO
         SET result = result * i;
         SET i = i + 1;
     END WHILE;
 END$$
 DELIMITER ;


CALL CalculateFactorial(5, @factorial_result);
 SELECT @factorial_result AS Factorial;


-- 37. PL/SQL: Display the Fibonacci series up to n terms
 DELIMITER $$
 CREATE PROCEDURE FibonacciSeries(IN n INT)
 BEGIN
     DECLARE a INT DEFAULT 0;
     DECLARE b INT DEFAULT 1;
     DECLARE temp INT;
     DECLARE i INT DEFAULT 1;

     WHILE i <= n DO
         SELECT a;
         SET temp = a + b;
         SET a = b;
         SET b = temp;
         SET i = i + 1;
     END WHILE;
 END$$
 DELIMITER ;


CALL FibonacciSeries(10);


-- 38. PL/SQL: Reverse a given string
 DELIMITER $$
 CREATE PROCEDURE ReverseString(IN input_str VARCHAR(100), OUT reversed_str VARCHAR(100))
 BEGIN
     DECLARE len INT;
     DECLARE i INT DEFAULT 1;
     SET reversed_str = '';
     SET len = CHAR_LENGTH(input_str);
     WHILE i <= len DO
         SET reversed_str = CONCAT(SUBSTRING(input_str, i, 1), reversed_str);
         SET i = i + 1;
     END WHILE;
 END$$
 DELIMITER ;


CALL ReverseString('OpenAI', @reversed_output);
 SELECT @reversed_output AS ReversedString;


-- 39. PL/SQL: Check if a number is prime
 DELIMITER $$
 CREATE PROCEDURE CheckPrime(IN num INT, OUT is_prime BOOLEAN)
 BEGIN
     DECLARE i INT DEFAULT 2;
     SET is_prime = TRUE;
     WHILE i <= SQRT(num) DO
         IF num MOD i = 0 THEN
             SET is_prime = FALSE;
```

```
        END IF;
        SET i = i + 1;
    END WHILE;
 END$$
 DELIMITER ;


CALL CheckPrime(17, @is_prime_result);
 SELECT @is_prime_result AS IsPrime;


-- 40. PL/SQL: Sum of all digits in a number
 DELIMITER $$
 CREATE PROCEDURE SumDigits(IN num INT, OUT digit_sum INT)
 BEGIN
    SET digit_sum = 0;
    WHILE num > 0 DO
        SET digit_sum = digit_sum + (num MOD 10);
        SET num = num DIV 10;
    END WHILE;
 END$$
 DELIMITER ;


CALL SumDigits(12345, @digit_sum_result);
 SELECT @digit_sum_result AS DigitSum;


-- 41


DELIMITER $$
 CREATE PROCEDURE DisplaySalaries()
 BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE emp_name VARCHAR(50);
    DECLARE emp_salary DECIMAL(10, 2);
    DECLARE cur CURSOR FOR SELECT name, salary FROM employees;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO emp_name, emp_salary;
        IF done THEN
            LEAVE read_loop;
        END IF;
        SELECT emp_name AS EmployeeName, emp_salary AS Salary;
    END LOOP;
    CLOSE cur;
 END$$
 DELIMITER ;

 CALL DisplaySalaries();



-- 42



DELIMITER $$
 CREATE PROCEDURE AvgSalaryByDepartment()
 BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE dept_id INT;
    DECLARE avg_salary DECIMAL(10, 2);
    DECLARE cur CURSOR FOR SELECT department_id, AVG(salary) FROM employees GROUP BY
department_id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO dept_id, avg_salary;
        IF done THEN
            LEAVE read_loop;
        END IF;
        SELECT dept_id AS DepartmentID, avg_salary AS AverageSalary;
    END LOOP;
    CLOSE cur;
 END$$
 DELIMITER ;

 CALL AvgSalaryByDepartment();



-- 43
 -- CREATE TRIGGER UpdateLastModified
 -- BEFORE UPDATE ON employees
 -- FOR EACH ROW
 -- BEGIN
 -- END;



-- 44
 -- CREATE TRIGGER PreventDepartmentDelete
 -- BEFORE DELETE ON departments
 -- FOR EACH ROW
 -- BEGIN
 --     IF (SELECT COUNT(*) FROM employees WHERE department_id = OLD.department_id) > 0 THEN
 --         SIGNAL SQLSTATE '45000';
 --



-- 44
 -- CREATE TABLE salary_changes_log (
 --     log_id INT AUTO_INCREMENT PRIMARY KEY,
 --     employee_id INT,
 --     old_salary DECIMAL(10, 2),
 --     new_salary DECIMAL(10, 2),
 --     change_date DATETIME
 -- );
 --



-- CREATE TRIGGER LogSalaryChanges
 -- AFTER UPDATE ON employees
 -- FOR EACH ROW
 -- BEGIN
 --     IF OLD.salary <> NEW.salary THEN
 --         INSERT INTO salary_changes_log (employee_id, old_salary, new_salary, change_date)
 -- VALUES (NEW.id, OLD.salary, NEW.salary, NOW());
 --     END IF;
 -- END;


-- 46
 -- CREATE TRIGGER PreventNegativeSalary
 -- BEFORE INSERT OR UPDATE ON employees
 -- FOR EACH ROW
 -- BEGIN
 --     IF NEW.salary < 0 THEN
 --         SIGNAL SQLSTATE '45000'
 --         SET MESSAGE_TEXT = 'Salary cannot be negative.';
 --     END IF;
 -- END;
```