PL/SQL - Assignment

```
create database PLSQL;
use PLSQL;

CREATE TABLE EMPLOYEES (
    EMP_ID INT PRIMARY KEY,
    EMP_NAME VARCHAR(100),
    DEPARTMENT VARCHAR(50),
    SALARY DECIMAL(10, 2)
);
-- ------------------------- 1 -----------------------------
DELIMITER $$
CREATE PROCEDURE insert_employee (
    IN p_emp_id INT,
    IN p_emp_name VARCHAR(100),
    IN p_department VARCHAR(50),
    IN p_salary DECIMAL(10, 2)
)
BEGIN
    INSERT INTO EMPLOYEES (EMP_ID, EMP_NAME, DEPARTMENT, SALARY)
    VALUES (p_emp_id, p_emp_name, p_department, p_salary);
END $$
DELIMITER ;

TRUNCATE TABLE EMPLOYEES;

CALL insert_employee(1, 'Sathiya', 'It', 20000);
CALL insert_employee(2, 'Selvi', 'Sales', 7000);
CALL insert_employee(3, 'Raj', 'Manager', 9000);
CALL insert_employee(4, 'Ram', 'Tester', 50000);
```

```
select * from EMPLOYEES;


-- -------------------------- 2 -------------------------
DELIMITER $$
CREATE PROCEDURE update_salary (IN p_emp_id INT)
BEGIN
   DECLARE current_salary DECIMAL(10, 2);


   -- Get the current salary of the employee
   SELECT SALARY INTO current_salary
   FROM EMPLOYEES
   WHERE EMP_ID = p_emp_id;


   -- Update the salary based on the current salary
   IF current_salary < 5000 THEN
      UPDATE EMPLOYEES
      SET SALARY = current_salary * 1.10
      WHERE EMP_ID = p_emp_id;
   ELSEIF current_salary BETWEEN 5000 AND 10000 THEN
      UPDATE EMPLOYEES
      SET SALARY = current_salary * 1.075
      WHERE EMP_ID = p_emp_id;
   ELSE
      UPDATE EMPLOYEES
      SET SALARY = current_salary * 1.05
      WHERE EMP_ID = p_emp_id;
   END IF;
END $$
DELIMITER ;


SET SQL_SAFE_UPDATES = 1;
```

```
CALL update_salary(1);

CALL update_salary(2);

SELECT * FROM EMPLOYEES;


-- ----------------------- 3 ------------------------

DELIMITER $$

CREATE PROCEDURE display_employee_names()

BEGIN

   DECLARE done INT DEFAULT 0;

   DECLARE emp_name VARCHAR(100);


   -- Declare cursor

   DECLARE emp_cursor CURSOR FOR

      SELECT EMP_NAME FROM EMPLOYEES;


   -- Declare continue handler

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;


   -- Open cursor

   OPEN emp_cursor;


   -- Fetch each row and display employee name

   read_loop: LOOP

      FETCH emp_cursor INTO emp_name;

      IF done THEN

         LEAVE read_loop;

      END IF;

      -- Display employee name using a SELECT statement to mimic printing

      SELECT emp_name AS Employee_Name;

   END LOOP;
```

```
    -- Close cursor
    CLOSE emp_cursor;
END $$
DELIMITER ;

CALL display_employee_names();

-- ------------------------ 4 -----------------------
CREATE VIEW high_salary_employees AS
SELECT EMP_ID, EMP_NAME, DEPARTMENT, SALARY
FROM EMPLOYEES
WHERE SALARY > 10000;

SELECT * FROM high_salary_employees;

-- ------------------------ 5 -------------------------
DELIMITER $$
CREATE FUNCTION Calculate_bonus (p_salary DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE bonus DECIMAL(10, 2);

    IF p_salary < 5000 THEN
        SET bonus = p_salary * 0.10;
    ELSEIF p_salary BETWEEN 5000 AND 10000 THEN
        SET bonus = p_salary * 0.075;
    ELSE
        SET bonus = p_salary * 0.05;
    END IF;
    RETURN bonus;
END $$
DELIMITER ;

SELECT EMP_ID, EMP_NAME, SALARY, Calculate_bonus(SALARY) AS BONUS
FROM EMPLOYEES;

-- ---------------------- 6 ----------------------
CREATE TABLE Employee_Log (
    Log_Id INT AUTO_INCREMENT PRIMARY KEY,
    EMP_ID INT,
    EMP_NAME VARCHAR(100),
    DEPARTMENT VARCHAR(50),
    SALARY DECIMAL(10, 2),
    INSERTION_TIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER $$
CREATE TRIGGER log_employee_insert
AFTER INSERT ON EMPLOYEES
FOR EACH ROW
BEGIN
    INSERT INTO Employee_Log (EMP_ID, EMP_NAME, DEPARTMENT, SALARY)
    VALUES (NEW.EMP_ID, NEW.EMP_NAME, NEW.DEPARTMENT, NEW.SALARY);
END $$
DELIMITER ;

SELECT * FROM Employee_Log;

-- ----------------------- 7 ----------------------
CREATE TABLE customers (
    customerid INT PRIMARY KEY,
    customer_name VARCHAR(100)
);
ALTER TABLE customers
ADD credit_limit DECIMAL(10, 2);

CREATE TABLE orders (
    orderid INT PRIMARY KEY,
    customerid INT,
    status VARCHAR(50),
    salesmanid INT,
    order_date DATE,
    FOREIGN KEY (customerid) REFERENCES customers(customerid)
);
SET SQL_SAFE_UPDATES = 0;
UPDATE customers
SET credit_limit = 0;

CREATE TABLE Order_Items (
    orderid INT,
    itemid INT,
    productid INT,
    quantity INT,
    unit_price DECIMAL(10, 2),
    PRIMARY KEY (orderid, itemid),
    FOREIGN KEY (orderid) REFERENCES orders(orderid)
);

INSERT INTO customers (customerid, customer_name) VALUES
(1, 'Sathiya'),
(2, 'Sathish'),
(3, 'Makesh');

INSERT INTO orders (orderid, customerid, status, salesmanid, order_date) VALUES
(1, 1, 'Shipped', 101, '2023-01-15'),
(2, 2, 'Pending', 102, '2023-02-21'),
(3, 1, 'Pending', 103, '2023-03-05'),
(4, 3, 'Shipped', 101, '2023-05-16');

INSERT INTO Order_Items (orderid, itemid, productid, quantity, unit_price) VALUES
(1, 1, 1001, 2, 500.00),
(1, 2, 1002, 1, 1500.00),
(2, 1, 1001, 3, 500.00),
(3, 1, 1003, 4, 250.00),
(4, 1, 1002, 2, 1500.00);

CREATE VIEW sales_revenues_by_customers AS
SELECT
   o.customerid,
   SUM(oi.quantity * oi.unit_price) AS total_sales_revenue,
   SUM(oi.quantity * oi.unit_price) * 0.05 AS credit
FROM
   orders o
JOIN
   Order_Items oi ON o.orderid = oi.orderid
GROUP BY
   o.customerid;

DELIMITER //
CREATE PROCEDURE update_credit_limits()
BEGIN
   DECLARE v_budget DECIMAL(10, 2) DEFAULT 1000000;
   DECLARE v_remaining_budget DECIMAL(10, 2) DEFAULT 1000000;
   DECLARE v_credit_limit DECIMAL(10, 2);
   DECLARE v_customerid INT;
   DECLARE v_total_sales_revenue DECIMAL(10, 2);
   DECLARE done INT DEFAULT 0;  -- Declare done as a local variable

   DECLARE customer_cursor CURSOR FOR
       SELECT customerid, total_sales_revenue
       FROM sales_revenues_by_customers
       ORDER BY total_sales_revenue DESC;

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   UPDATE customers SET credit_limit = 0;

   OPEN customer_cursor;

   read_loop: LOOP
       FETCH customer_cursor INTO v_customerid, v_total_sales_revenue;

       IF done THEN
           LEAVE read_loop;
       END IF;

       SET v_credit_limit = v_total_sales_revenue * 0.05;

       IF v_credit_limit > v_remaining_budget THEN
           SET v_credit_limit = v_remaining_budget;
       END IF;

       UPDATE customers
```

```
        SET credit_limit = v_credit_limit
        WHERE customerid = v_customerid;
        SET v_remaining_budget = v_remaining_budget - v_credit_limit;

        IF v_remaining_budget <= 0 THEN
            LEAVE read_loop;
        END IF;
    END LOOP;

    -- Close the cursor
    CLOSE customer_cursor;
END //
DELIMITER ;

CALL update_credit_limits();

-- ------------------------------8-----------------------------
CREATE TABLE employee (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    email VARCHAR(25),
    phone_number VARCHAR(15),
    hire_date DATE,
    job_id VARCHAR(25),
    salary INT,
    commission_pct DECIMAL(5,2),
    manager_id INT,
    department_id INT
);
```

```
INSERT INTO employee (employee_id, first_name, last_name, email, phone_number, hire_date,
job_id, salary, commission_pct, manager_id, department_id)

VALUES

(1, 'sathiya', 'banu', 'sathiya@gmail..com', '123-456-7890', '2020-07-19', 'IT_PROG', 50000, NULL,
101, 10),

(2, 'siva', 'kumar', 'siva@gmail.com', '987-654-3210', '2019-03-23', 'HR_REP', 35000, NULL, 102, 20),

(3, 'abi', 'nithi', 'abi@gmail.com', '456-789-0123', '2021-06-30', 'FIN_ANALYST', 60000, 0.10, 103, 30);


DELIMITER //

DROP PROCEDURE IF EXISTS display_employee_info;


DELIMITER //

CREATE PROCEDURE display_employee_info ()

BEGIN

   DECLARE v_employee_id INT;

   DECLARE v_first_name VARCHAR(25);

   DECLARE v_last_name VARCHAR(25);

   DECLARE v_email VARCHAR(25);

   DECLARE v_phone_number VARCHAR(15);

   DECLARE v_hire_date DATE;

   DECLARE v_job_id VARCHAR(25);

   DECLARE v_salary INT;

   DECLARE v_commission_pct DECIMAL(5,2);

   DECLARE v_manager_id INT;

   DECLARE v_department_id INT;


   -- Use implicit cursor to select employee information

   SELECT employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary,
commission_pct, manager_id, department_id

   INTO v_employee_id, v_first_name, v_last_name, v_email, v_phone_number, v_hire_date,
v_job_id, v_salary, v_commission_pct, v_manager_id, v_department_id

   FROM employees
```

```
    WHERE employee_id = 1;

    SELECT 'Employee ID: ', v_employee_id;
    SELECT 'First Name: ', v_first_name;
    SELECT 'Last Name: ', v_last_name;
    SELECT 'Email: ', v_email;
    SELECT 'Phone Number: ', v_phone_number;
    SELECT 'Hire Date: ', v_hire_date;
    SELECT 'Job ID: ', v_job_id;
    SELECT 'Salary: ', v_salary;
    SELECT 'Commission Pct: ', v_commission_pct;
    SELECT 'Manager ID: ', v_manager_id;
    SELECT 'Department ID: ', v_department_id;
END //
DELIMITER ;

CALL display_employee_info();

-- ------------------------------ 9 ------------------------------
DROP PROCEDURE IF EXISTS display_low_salary_employees;

DELIMITER //
CREATE PROCEDURE display_low_salary_employees(IN max_salary INT)
BEGIN
    -- Declare variables to hold the fetched data
    DECLARE v_first_name VARCHAR(25);
    DECLARE v_last_name VARCHAR(25);
    DECLARE v_salary INT;
    DECLARE done INT DEFAULT 0;

    -- Declare a cursor to fetch employee names and salaries
    DECLARE cur_employee CURSOR FOR
        SELECT first_name, last_name, salary
        FROM employees
        WHERE salary < max_salary;

    -- Declare a handler to set done to 1 when no more rows are found
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Open the cursor
    OPEN cur_employee;

    -- Loop through each row fetched by the cursor
    read_loop: LOOP
        -- Fetch the data into variables
        FETCH cur_employee INTO v_first_name, v_last_name, v_salary;

        -- Exit the loop if no more rows are found
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Display the fetched data
        SELECT CONCAT('Name: ', v_first_name, ' ', v_last_name, ' - Salary: ', v_salary) AS
Employee_Info;
    END LOOP;

    -- Close the cursor
    CLOSE cur_employee;
END //

DELIMITER ;
```

```
SET SQL_SAFE_UPDATES = 0;


CALL display_low_salary_employees(50000);


-- ---------------------- 10 -----------------------

DELIMITER //

CREATE TRIGGER check_duplicate_email

BEFORE INSERT  ON employees

FOR EACH ROW

BEGIN

   DECLARE email_count INT;


   -- Check for duplicate email in the table

   SELECT COUNT(*)

   INTO email_count

   FROM employees

   WHERE email = NEW.email AND employee_id != NEW.employee_id;


   -- Raise an exception if a duplicate email is found

   IF email_count > 0 THEN

      SIGNAL SQLSTATE '45000'

      SET MESSAGE_TEXT = 'Duplicate email address found. Each employee must have a unique email
address.';

   END IF;

END //

DELIMITER ;


INSERT INTO employees (employee_id, first_name, last_name, email, phone_number, hire_date,
job_id, salary, commission_pct, manager_id, department_id)

VALUES (4, 'John', 's', 'john.doe@example.com', '555-1234', '2024-07-19', 'IT_PROG', 60000, NULL,
NULL, 60);
```

```
INSERT INTO employees (employee_id, first_name, last_name, email, phone_number, hire_date,
job_id, salary, commission_pct, manager_id, department_id)

VALUES (5, 'ms', 'dhoni', 'john.doe@example.com', '555-5678', '2024-07-19', 'IT_PROG', 70000,
NULL, NULL, 60);


-- ------------------------ 11 --------------------------

DROP PROCEDURE IF EXISTS get_employees_by_salary;


DELIMITER //

CREATE PROCEDURE get_employees_by_salary(IN min_salary DECIMAL(10, 2), IN max_salary
DECIMAL(10, 2))

BEGIN

    -- Select employees whose salary is between min_salary and max_salary

    SELECT

        employee_id, first_name,  last_name, email,

        phone_number,  hire_date,  job_id, salary,

        commission_pct, manager_id, department_id

    FROM  employees

    WHERE

        salary BETWEEN min_salary AND max_salary;

END //

DELIMITER ;


CALL get_employees_by_salary(30000, 70000);


-- -------------------------- 12 ------------------------

CREATE TABLE employeesTable (

    employee_id INT PRIMARY KEY,

    first_name VARCHAR(25),

    last_name VARCHAR(25),
```

```sql
    email_id VARCHAR(50),
    phone_number VARCHAR(15),
    join_date DATE,
    job_id VARCHAR(25),
    salary DECIMAL(10, 2)
);

INSERT INTO employeesTable (employee_id, first_name, last_name, email_id, phone_number,
join_date, job_id, salary)
VALUES
(100, 'ABC', 'DEF', 'abef', '9876543210', '2020-06-06', 'AD_PRES', 24000.00),
(101, 'GHI', 'JKL', 'ghkl', '9876543211', '2021-02-08', 'AD_VP', 17000.00),
(102, 'MNO', 'PQR', 'mnqr', '9876543212', '2016-05-14', 'AD_VP', 17000.00),
(103, 'STU', 'VWX', 'stwx', '9876543213', '2019-06-24', 'IT_PROG', 9000.00);

DELIMITER //

CREATE PROCEDURE increment_salary(employee_id INT, increment_amount DECIMAL(10,2))
BEGIN
    UPDATE employeesTable
    SET salary = salary + increment_amount
    WHERE employee_id = employee_id;
END //

DELIMITER ;

CALL increment_salary(102, 1000);

SELECT * FROM employeesTable WHERE employee_id = 102;
```