TITLE: PL/SQL Assignment

AUTHOR: Preetha I

CREATED DATE: 14-07-2024

1: Create a Procedure to Insert Employee Data

```
CREATE OR REPLACE PROCEDURE insert_employee (
    p_emp_id     IN NUMBER,
    p_emp_name   IN VARCHAR2,
    p_department IN VARCHAR2,
    p_salary     IN NUMBER
) AS
BEGIN
    INSERT INTO EMPLOYEES (EMP_ID, EMP_NAME, DEPARTMENT, SALARY)
    VALUES (p_emp_id, p_emp_name, p_department, p_salary);

    COMMIT;
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Error: Employee ID already exists.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

2: Create a Procedure to Update Employee Salary

```
CREATE OR REPLACE PROCEDURE update_salary (
    p_emp_id IN NUMBER
) AS
    v_current_salary EMPLOYEES.SALARY%TYPE;
```

```
    v_new_salary EMPLOYEES.SALARY%TYPE;
BEGIN
    SELECT SALARY INTO v_current_salary
    FROM EMPLOYEES
    WHERE EMP_ID = p_emp_id;
    IF v_current_salary < 5000 THEN
        v_new_salary := v_current_salary * 1.10;
    ELSIF v_current_salary BETWEEN 5000 AND 10000 THEN
        v_new_salary := v_current_salary * 1.075;
    ELSE
        v_new_salary := v_current_salary * 1.05;
    END IF;
    UPDATE EMPLOYEES
    SET SALARY = v_new_salary
    WHERE EMP_ID = p_emp_id;

    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: Employee ID not found.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

3: Use a Cursor to Display Employee Names

```
DECLARE
    CURSOR emp_cursor IS
        SELECT EMP_NAME FROM EMPLOYEES;
```

```
    v_emp_name EMPLOYEES.EMP_NAME%TYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_emp_name;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_emp_name);
    END LOOP;
    CLOSE emp_cursor;
END;
```

4: Create a View for Employees with High Salary

```
CREATE OR REPLACE VIEW high_salary_employees AS
SELECT EMP_ID, EMP_NAME, DEPARTMENT, SALARY
FROM EMPLOYEES
WHERE SALARY > 10000;
```

5: Create a Function to Calculate Bonus

```
CREATE OR REPLACE FUNCTION calculate_bonus (
    p_salary IN NUMBER
) RETURN NUMBER IS
    v_bonus NUMBER;
BEGIN
    IF p_salary < 5000 THEN
        v_bonus := p_salary * 0.10;
    ELSIF p_salary BETWEEN 5000 AND 10000 THEN
        v_bonus := p_salary * 0.075;
    ELSE
```

```
        v_bonus := p_salary * 0.05;
    END IF;

    RETURN v_bonus;
END calculate_bonus;
```

6: Create a Trigger to Log Employee Insertions

```
CREATE OR REPLACE 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;
```

7: Consider the orders and order_items tables from the sample database.

A) Create a view that returns the sales revenues by customers. The values of the credit column are 5% of the total sales revenues.

```
CREATE OR REPLACE VIEW high_salary_employees AS
SELECT
    o.CUSTOMER_ID,
    SUM(oi.QUANTITY * oi.UNIT_PRICE) AS TOTAL_SALES,
    SUM(oi.QUANTITY * oi.UNIT_PRICE) * 0.05 AS CREDIT
FROM
    ORDERS o
JOIN
    ORDER_ITEMS oi ON o.ORDER_ID = oi.ORDER_ID
GROUP BY
```

```
    o.CUSTOMER_ID;
```

B) Write the PL/SQL query to develop an anonymous block which:

1. Reset the credit limits of all customers to zero.

2. Fetch customers sorted by sales in descending order and give them new credit limits from a budget of 1 million.

```
DECLARE
    CURSOR c_customers IS
        SELECT CUSTOMER_ID, TOTAL_SALES
        FROM high_salary_employees
        ORDER BY TOTAL_SALES DESC;

    v_budget NUMBER := 1000000;
    v_credit_limit NUMBER;
    v_customer_id ORDERS.CUSTOMER_ID%TYPE;
BEGIN
    -- Reset the credit limits of all customers to zero
    UPDATE high_salary_employees
    SET CREDIT = 0;

    -- Fetch customers sorted by sales in descending order and assign new credit limits
    FOR r_customer IN c_customers LOOP
        v_credit_limit := r_customer.TOTAL_SALES * 0.05;
        IF v_budget >= v_credit_limit THEN
            UPDATE high_salary_employees
            SET CREDIT = v_credit_limit
            WHERE CUSTOMER_ID = r_customer.CUSTOMER_ID;
            v_budget := v_budget - v_credit_limit;
```

```
      ELSE
         UPDATE high_salary_employees
         SET CREDIT = v_budget
         WHERE CUSTOMER_ID = r_customer.CUSTOMER_ID;
         EXIT;
      END IF;
   END LOOP;
END;
```

8: Write a program in PL/SQL to show the uses of implicit cursor without using any attribute.

```
DECLARE
   -- Variables to hold employee data
   v_employee_id    employees.employee_id%TYPE;
   v_first_name     employees.first_name%TYPE;
   v_last_name      employees.last_name%TYPE;
   v_email          employees.email%TYPE;
   v_phone_number   employees.phone_number%TYPE;
   v_hire_date      employees.hire_date%TYPE;
   v_job_id         employees.job_id%TYPE;
   v_salary         employees.salary%TYPE;
   v_commission_pct employees.commission_pct%TYPE;
   v_manager_id     employees.manager_id%TYPE;
   v_department_id  employees.department_id%TYPE;
BEGIN
   -- Fetching employee details for a specific employee_id
   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 = 101; -- Replace with the employee_id you want to query

    -- Display the employee details
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id);
    DBMS_OUTPUT.PUT_LINE('First Name: ' || v_first_name);
    DBMS_OUTPUT.PUT_LINE('Last Name: ' || v_last_name);
    DBMS_OUTPUT.PUT_LINE('Email: ' || v_email);
    DBMS_OUTPUT.PUT_LINE('Phone Number: ' || v_phone_number);
    DBMS_OUTPUT.PUT_LINE('Hire Date: ' || v_hire_date);
    DBMS_OUTPUT.PUT_LINE('Job ID: ' || v_job_id);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || v_salary);
    DBMS_OUTPUT.PUT_LINE('Commission PCT: ' || v_commission_pct);
    DBMS_OUTPUT.PUT_LINE('Manager ID: ' || v_manager_id);
    DBMS_OUTPUT.PUT_LINE('Department ID: ' || v_department_id);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employee found with the given ID.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Query returned more than one row.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/
```

9: Write a program in PL/SQL to create a cursor displays the name and salary of each employee in the EMPLOYEES table whose salary is less than that specified by a passed-in parameter value.

```
DECLARE
```

```
    -- Parameter for salary threshold
    p_salary_threshold employees.salary%TYPE := 50000;    -- Cursor definition
    CURSOR emp_cursor IS
        SELECT first_name, last_name, salary
        FROM employees
        WHERE salary < p_salary_threshold;

    -- Variables to hold cursor data
    v_first_name employees.first_name%TYPE;
    v_last_name  employees.last_name%TYPE;
    v_salary     employees.salary%TYPE;
BEGIN
    -- Open the cursor and loop through the results
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_first_name, v_last_name, v_salary;
        EXIT WHEN emp_cursor%NOTFOUND;
        -- Display employee details
        DBMS_OUTPUT.PUT_LINE('Name: ' || v_first_name || ' ' || v_last_name || ', Salary: ' ||
v_salary);
    END LOOP;

    -- Close the cursor
    CLOSE emp_cursor;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/
```

10: Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found.

```
CREATE OR REPLACE TRIGGER check_duplicate_email
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
DECLARE
   -- Variable to store the count of rows with the same email
   v_count INTEGER;
BEGIN
   -- Check for duplicate email
   SELECT COUNT(*)
   INTO v_count
   FROM employees
   WHERE email = :NEW.email
     AND employee_id != :NEW.employee_id; -- Exclude the current row during update

   -- Raise an exception if a duplicate is found
   IF v_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Duplicate email address found: ' || :NEW.email);
   END IF;
END;
/
```

11: Write a PL/SQL procedure for selecting some records from the database using some parameters as filters.

```
CREATE OR REPLACE PROCEDURE get_employees_by_salary (
   p_salary_threshold IN ib_employee.salary%TYPE
)
```

```
IS
   -- Cursor to fetch employees with a salary greater than or equal to the specified threshold
   CURSOR emp_cursor IS
      SELECT employee_id, first_name, last_name, salary
      FROM ib_employee
      WHERE salary >= p_salary_threshold;

   -- Variables to hold cursor data
   v_employee_id    ib_employee.employee_id%TYPE;
   v_first_name     ib_employee.first_name%TYPE;
   v_last_name      ib_employee.last_name%TYPE;
   v_salary         ib_employee.salary%TYPE;
BEGIN
   -- Open the cursor and loop through the results
   OPEN emp_cursor;
   LOOP
      FETCH emp_cursor INTO v_employee_id, v_first_name, v_last_name, v_salary;
      EXIT WHEN emp_cursor%NOTFOUND;
      -- Display employee details
      DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id);
      DBMS_OUTPUT.PUT_LINE('First Name: ' || v_first_name);
      DBMS_OUTPUT.PUT_LINE('Last Name: ' || v_last_name);
      DBMS_OUTPUT.PUT_LINE('Salary: ' || v_salary);
      DBMS_OUTPUT.PUT_LINE('-----------------------------');
   END LOOP;

   -- Close the cursor
   CLOSE emp_cursor;
```

```
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/
```

12: Write PL/SQL code block to increment the employee's salary by 1000 whose employee_id is 102.

```
BEGIN
   -- Update the salary for the employee with employee_id = 102
   UPDATE EMPLOYE
   SET SALARY = SALARY + 1000
   WHERE E_ID = 102;
   COMMIT;

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
      ROLLBACK;
END;
/
```