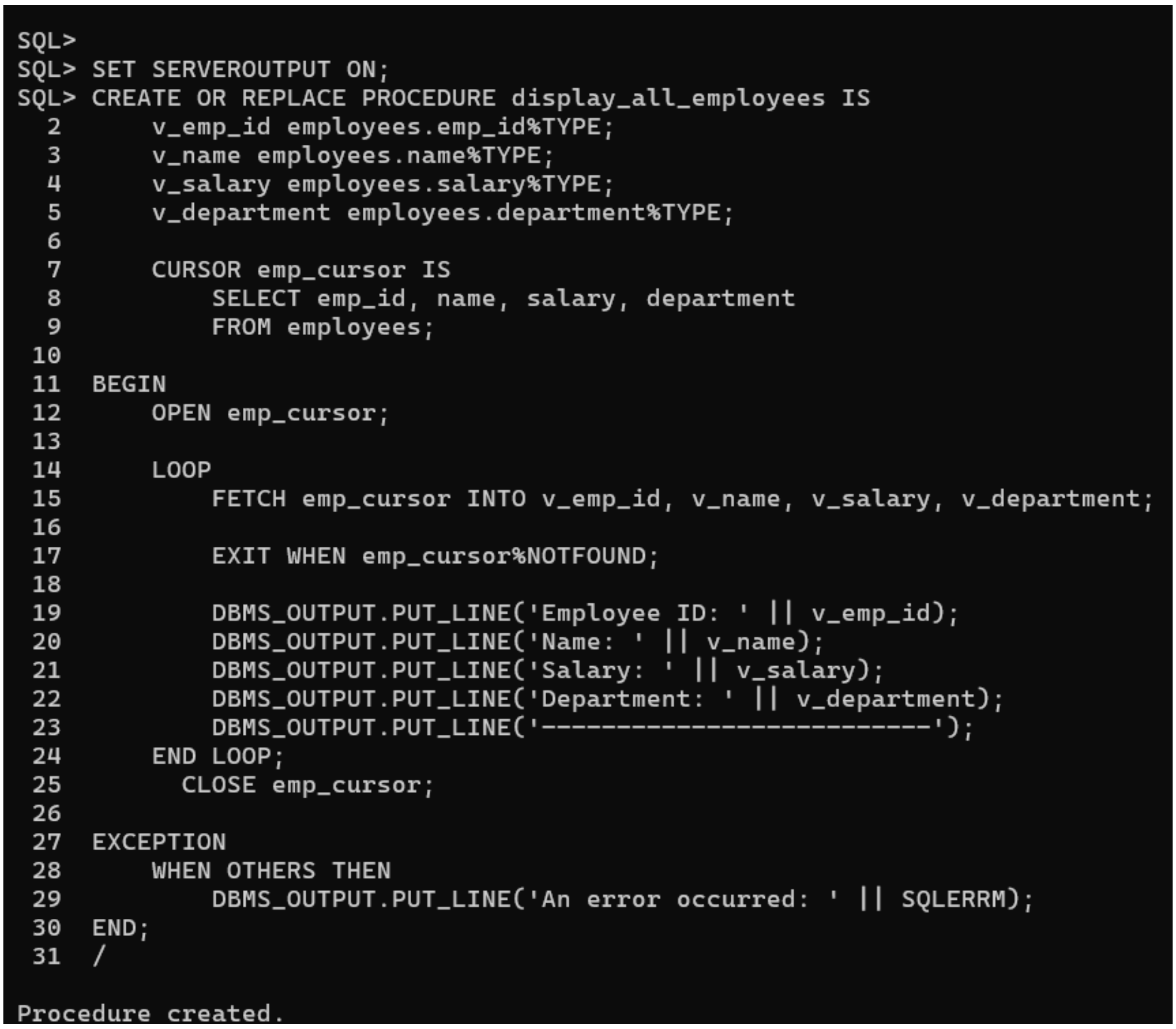
ORACLE ASSIGNMENT LAB– 10

Procedure

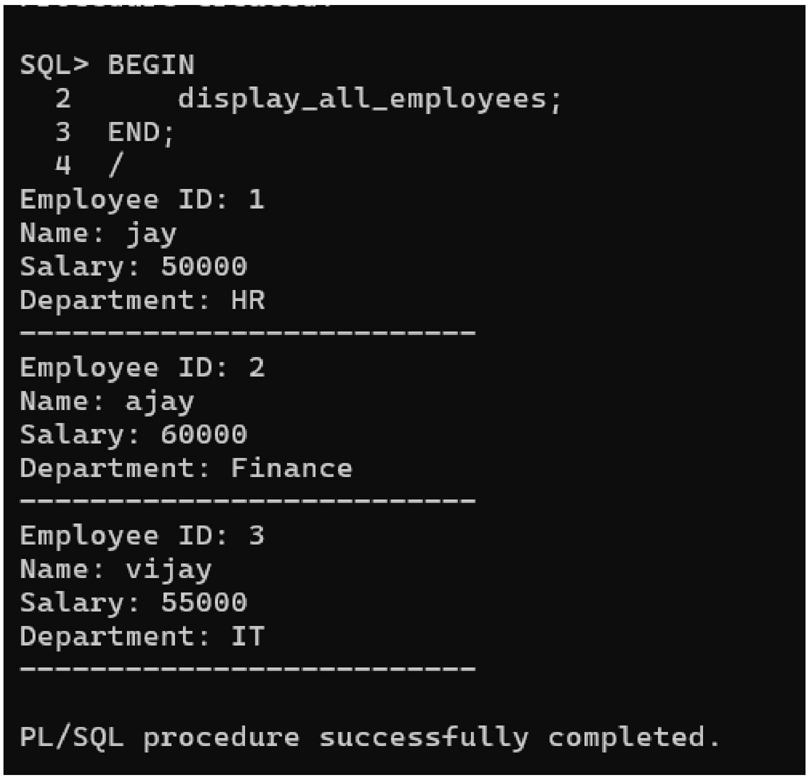
1. Write a procedure that will display all employee details.

- Write procedure



```
SQL>
SQL> SET SERVEROUTPUT ON;
SQL> CREATE OR REPLACE PROCEDURE display_all_employees IS
         v_emp_id employees.emp_id%TYPE;
         v_name employees.name%TYPE;
         v_salary employees.salary%TYPE;
         v_department employees.department%TYPE;

         CURSOR emp_cursor IS
             SELECT emp_id, name, salary, department
             FROM employees;

     BEGIN
         OPEN emp_cursor;

         LOOP
             FETCH emp_cursor INTO v_emp_id, v_name, v_salary, v_department;

             EXIT WHEN emp_cursor%NOTFOUND;

             DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_emp_id);
             DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
             DBMS_OUTPUT.PUT_LINE('Salary: ' || v_salary);
             DBMS_OUTPUT.PUT_LINE('Department: ' || v_department);
             DBMS_OUTPUT.PUT_LINE('--------------------------');
         END LOOP;
           CLOSE emp_cursor;

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

Procedure created.
```

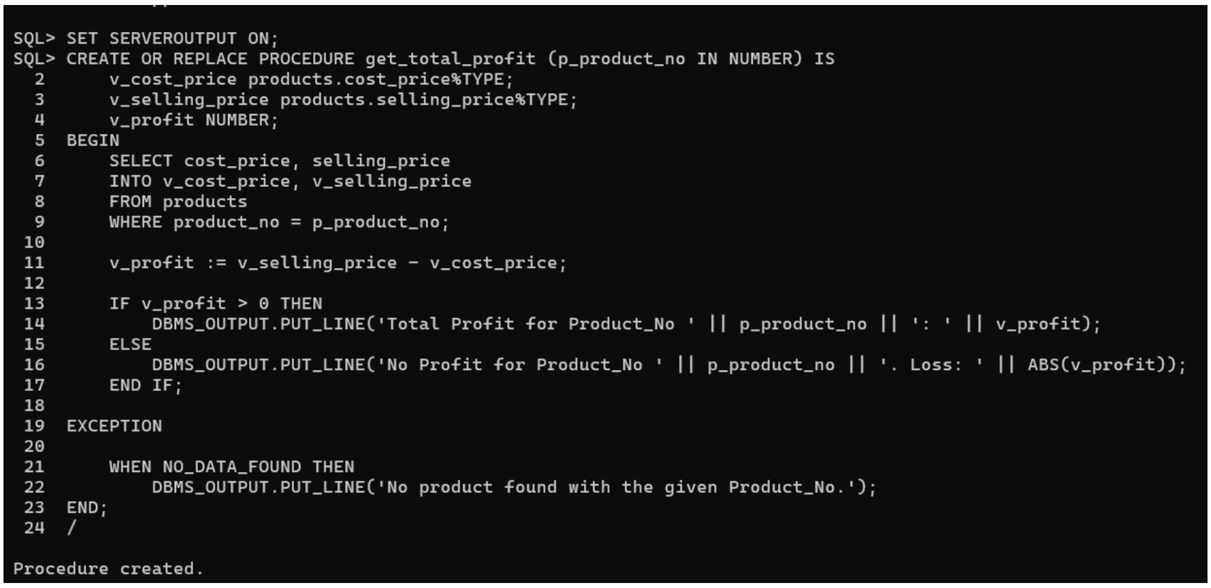
- Run procedure



```
SQL> BEGIN
        display_all_employees;
     END;
     /
Employee ID: 1
Name: jay
Salary: 50000
Department: HR
--------------------------
Employee ID: 2
Name: ajay
Salary: 60000
Department: Finance
--------------------------
Employee ID: 3
Name: vijay
Salary: 55000
Department: IT
--------------------------

PL/SQL procedure successfully completed.
```

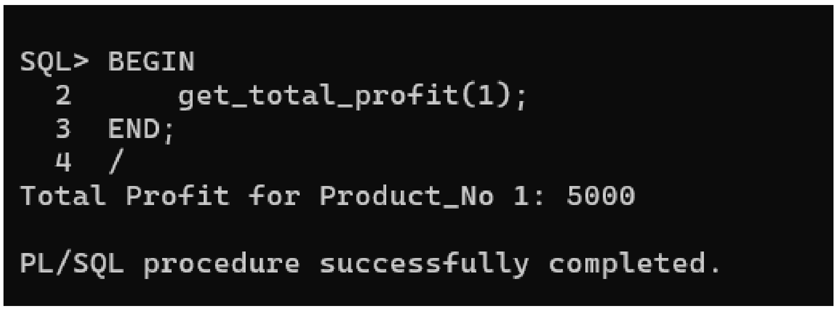
2. Write a procedure that will find out the total profit for the entered Product_No.

- Write procedure



```
SQL> SET SERVEROUTPUT ON;
SQL> CREATE OR REPLACE PROCEDURE get_total_profit (p_product_no IN NUMBER) IS
         v_cost_price products.cost_price%TYPE;
         v_selling_price products.selling_price%TYPE;
         v_profit NUMBER;
     BEGIN
         SELECT cost_price, selling_price
         INTO v_cost_price, v_selling_price
         FROM products
         WHERE product_no = p_product_no;

         v_profit := v_selling_price - v_cost_price;

         IF v_profit > 0 THEN
             DBMS_OUTPUT.PUT_LINE('Total Profit for Product_No ' || p_product_no || ': ' || v_profit);
         ELSE
             DBMS_OUTPUT.PUT_LINE('No Profit for Product_No ' || p_product_no || '. Loss: ' || ABS(v_profit));
         END IF;

     EXCEPTION

         WHEN NO_DATA_FOUND THEN
             DBMS_OUTPUT.PUT_LINE('No product found with the given Product_No.');
     END;
     /

Procedure created.
```

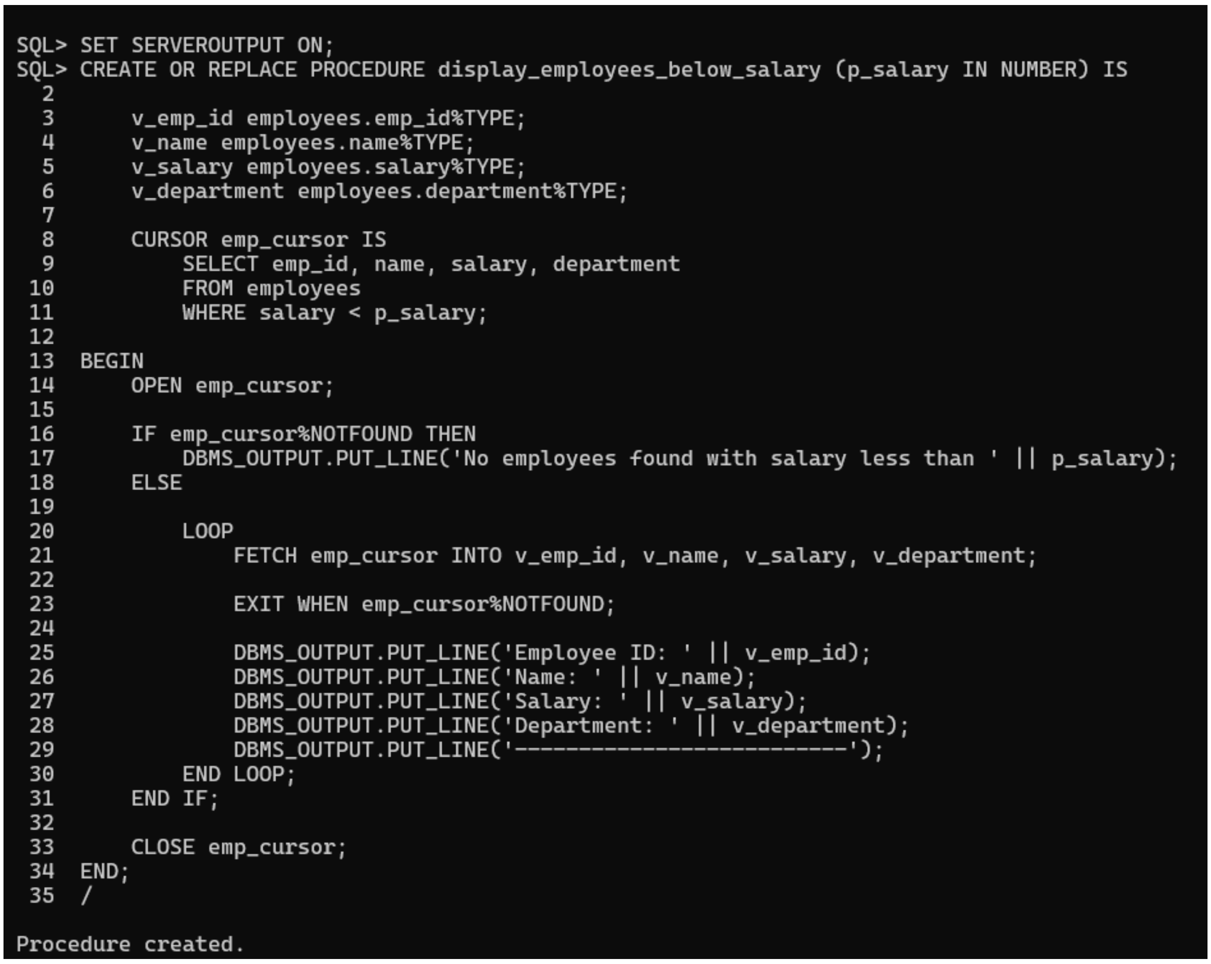
- Run procedure



```
SQL> BEGIN
  2      get_total_profit(1);
  3  END;
  4  /
Total Profit for Product_No 1: 5000

PL/SQL procedure successfully completed.
```

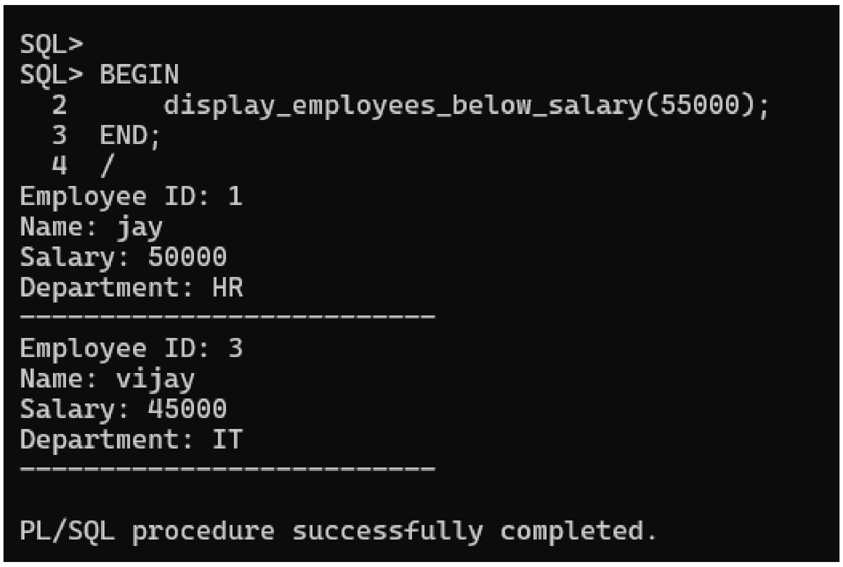
3. Write a procedure that will display employee details whose salary is less then entered salary by user.

- Write procedure



```
SQL> SET SERVEROUTPUT ON;
SQL> CREATE OR REPLACE PROCEDURE display_employees_below_salary (p_salary IN NUMBER) IS

    v_emp_id employees.emp_id%TYPE;
    v_name employees.name%TYPE;
    v_salary employees.salary%TYPE;
    v_department employees.department%TYPE;

    CURSOR emp_cursor IS
        SELECT emp_id, name, salary, department
        FROM employees
        WHERE salary < p_salary;

BEGIN
    OPEN emp_cursor;

    IF emp_cursor%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employees found with salary less than ' || p_salary);
    ELSE

        LOOP
            FETCH emp_cursor INTO v_emp_id, v_name, v_salary, v_department;

            EXIT WHEN emp_cursor%NOTFOUND;

            DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_emp_id);
            DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
            DBMS_OUTPUT.PUT_LINE('Salary: ' || v_salary);
            DBMS_OUTPUT.PUT_LINE('Department: ' || v_department);
            DBMS_OUTPUT.PUT_LINE('---------------------------');
        END LOOP;
    END IF;

    CLOSE emp_cursor;
END;
/

Procedure created.
```

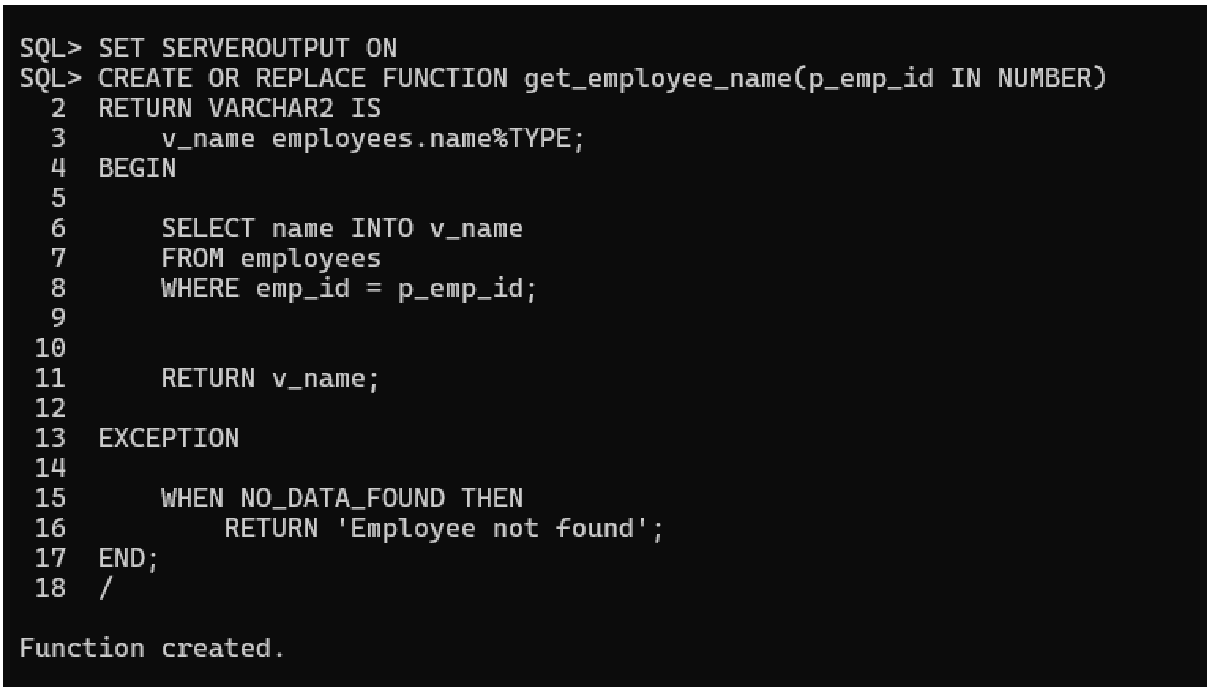
- Run procedure



```
SQL>
SQL> BEGIN
         display_employees_below_salary(55000);
     END;
     /
Employee ID: 1
Name: jay
Salary: 50000
Department: HR
--------------------------
Employee ID: 3
Name: vijay
Salary: 45000
Department: IT
--------------------------

PL/SQL procedure successfully completed.
```

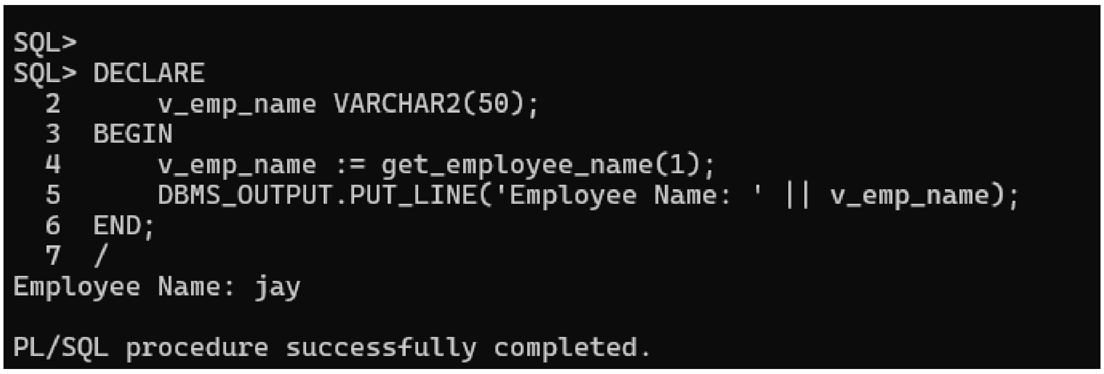
4. Write a function that will accept employee number and display employee name.

- Write function



```
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE FUNCTION get_employee_name(p_emp_id IN NUMBER)
     RETURN VARCHAR2 IS
         v_name employees.name%TYPE;
     BEGIN

         SELECT name INTO v_name
         FROM employees
         WHERE emp_id = p_emp_id;


         RETURN v_name;

     EXCEPTION

         WHEN NO_DATA_FOUND THEN
             RETURN 'Employee not found';
     END;
     /

Function created.
```

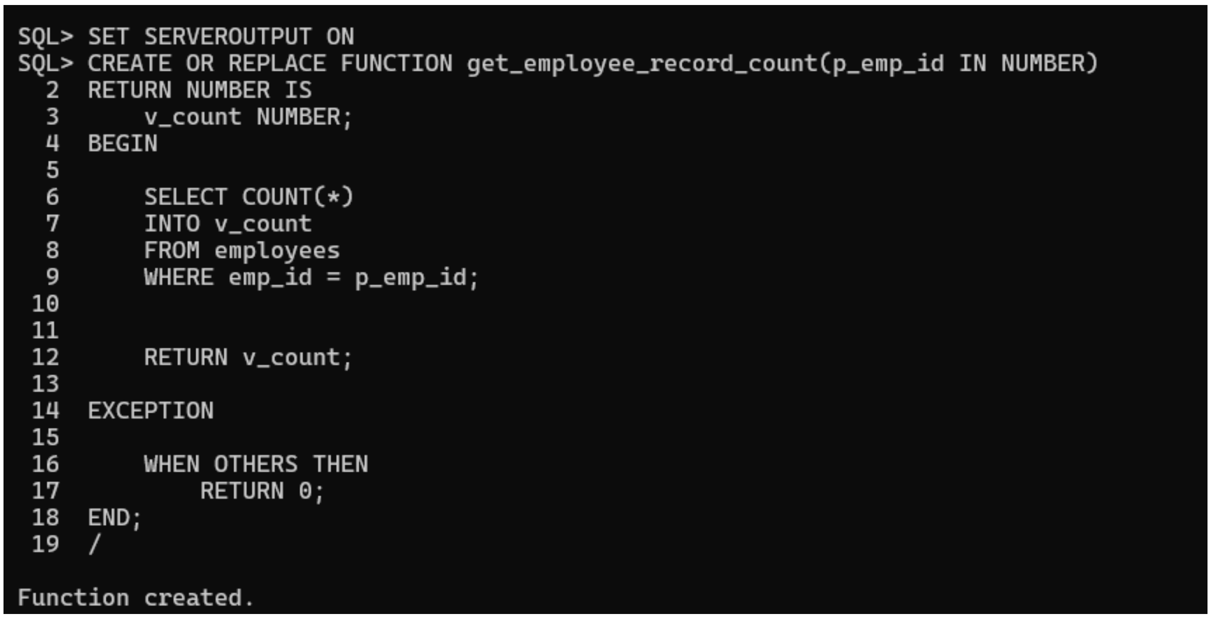
- Run function



```
SQL>
SQL> DECLARE
         v_emp_name VARCHAR2(50);
     BEGIN
         v_emp_name := get_employee_name(1);
         DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_emp_name);
     END;
     /
Employee Name: jay

PL/SQL procedure successfully completed.
```

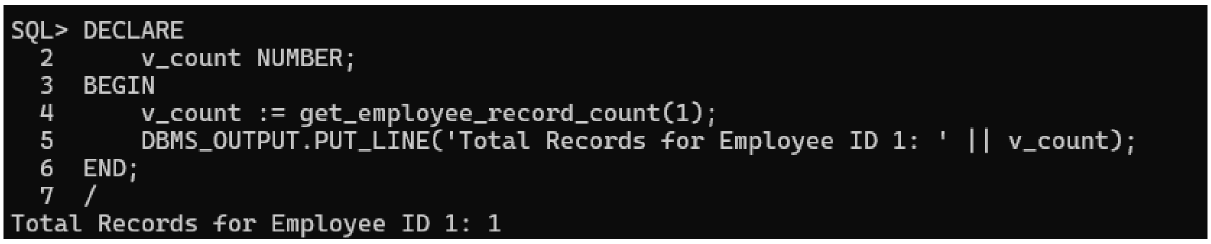
5. Write a function that will accept employee number and display total number of records exist for the employee number.

- Write function



```
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE FUNCTION get_employee_record_count(p_emp_id IN NUMBER)
     RETURN NUMBER IS
         v_count NUMBER;
     BEGIN

         SELECT COUNT(*)
         INTO v_count
         FROM employees
         WHERE emp_id = p_emp_id;


         RETURN v_count;

     EXCEPTION

         WHEN OTHERS THEN
             RETURN 0;
     END;
     /

Function created.
```

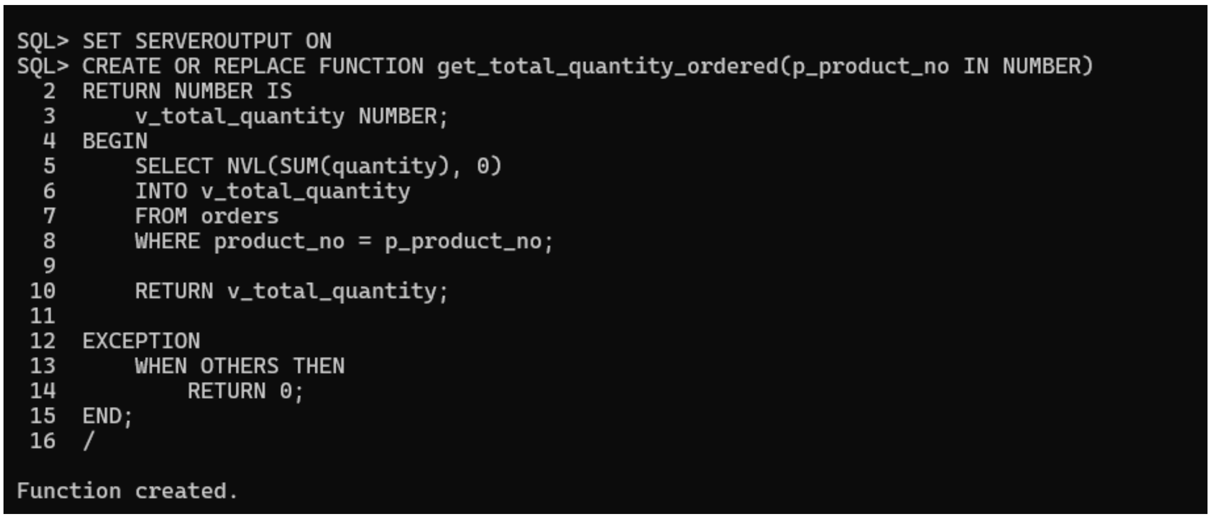
- Run function



```
SQL> DECLARE
         v_count NUMBER;
     BEGIN
         v_count := get_employee_record_count(1);
         DBMS_OUTPUT.PUT_LINE('Total Records for Employee ID 1: ' || v_count);
     END;
     /
Total Records for Employee ID 1: 1
```

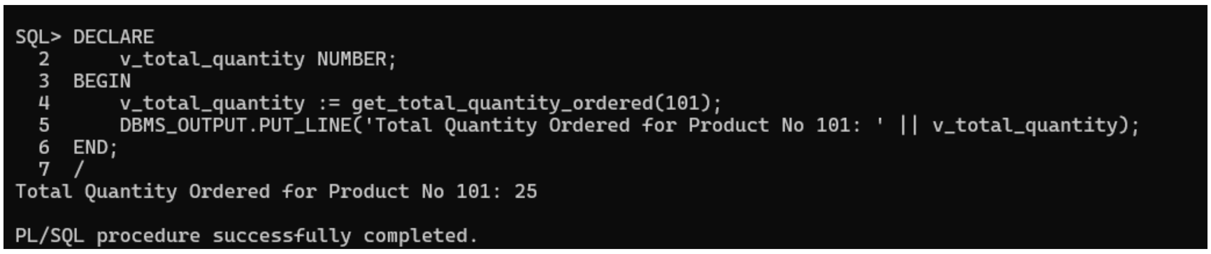
6. Write a function that will accept Product_No and find out total quantity order.

- Write function



```
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE FUNCTION get_total_quantity_ordered(p_product_no IN NUMBER)
     RETURN NUMBER IS
         v_total_quantity NUMBER;
     BEGIN
         SELECT NVL(SUM(quantity), 0)
         INTO v_total_quantity
         FROM orders
         WHERE product_no = p_product_no;

         RETURN v_total_quantity;

     EXCEPTION
         WHEN OTHERS THEN
             RETURN 0;
     END;
     /

Function created.
```

- Run function

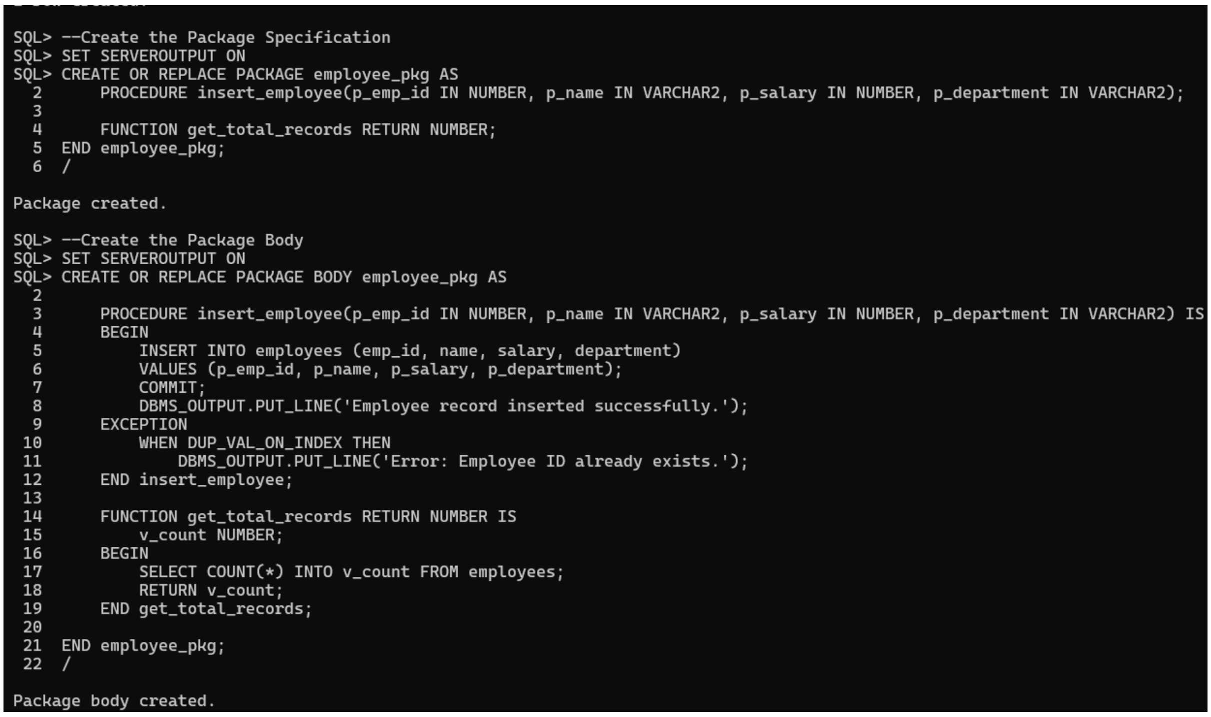


```
SQL> DECLARE
         v_total_quantity NUMBER;
     BEGIN
         v_total_quantity := get_total_quantity_ordered(101);
         DBMS_OUTPUT.PUT_LINE('Total Quantity Ordered for Product No 101: ' || v_total_quantity);
     END;
     /
Total Quantity Ordered for Product No 101: 25

PL/SQL procedure successfully completed.
```

PACKAGE PROGRAMS

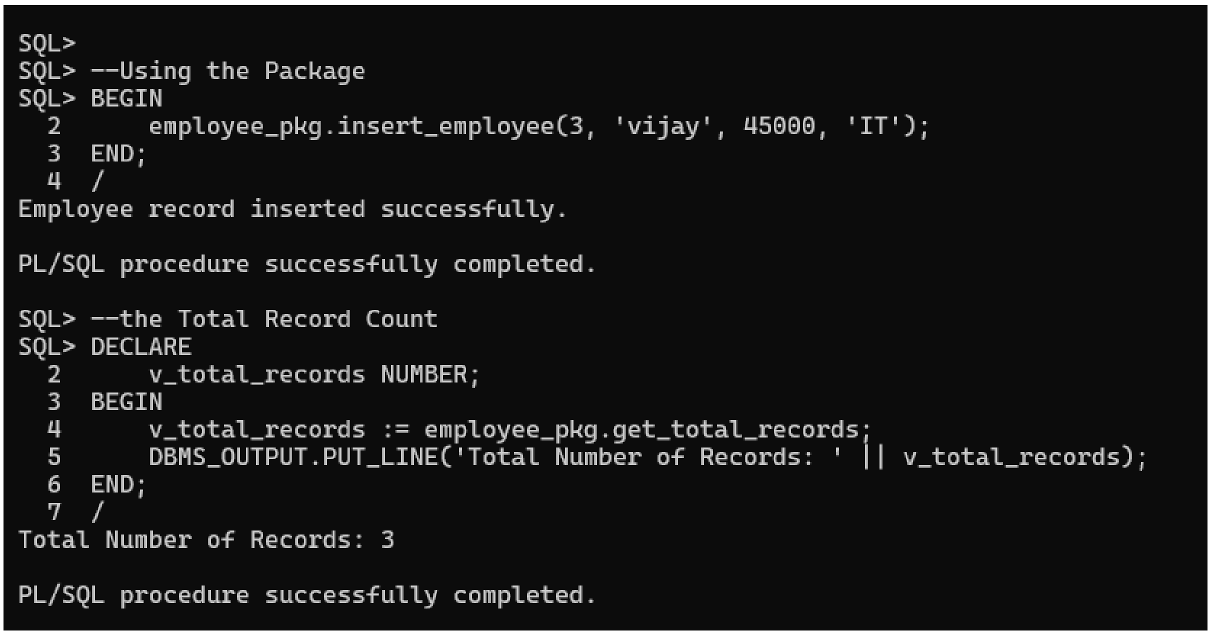
1. Create a package that will use procedure to insert a record in Employee table, and function to display total number of records available in an Employee table.

- Write package specification and package body



```
SQL> --Create the Package Specification
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PACKAGE employee_pkg AS
         PROCEDURE insert_employee(p_emp_id IN NUMBER, p_name IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2);

         FUNCTION get_total_records RETURN NUMBER;
     END employee_pkg;
     /

Package created.

SQL> --Create the Package Body
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PACKAGE BODY employee_pkg AS

         PROCEDURE insert_employee(p_emp_id IN NUMBER, p_name IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2) IS
         BEGIN
             INSERT INTO employees (emp_id, name, salary, department)
             VALUES (p_emp_id, p_name, p_salary, p_department);
             COMMIT;
             DBMS_OUTPUT.PUT_LINE('Employee record inserted successfully.');
         EXCEPTION
             WHEN DUP_VAL_ON_INDEX THEN
                 DBMS_OUTPUT.PUT_LINE('Error: Employee ID already exists.');
         END insert_employee;

         FUNCTION get_total_records RETURN NUMBER IS
             v_count NUMBER;
         BEGIN
             SELECT COUNT(*) INTO v_count FROM employees;
             RETURN v_count;
         END get_total_records;

     END employee_pkg;
     /

Package body created.
```

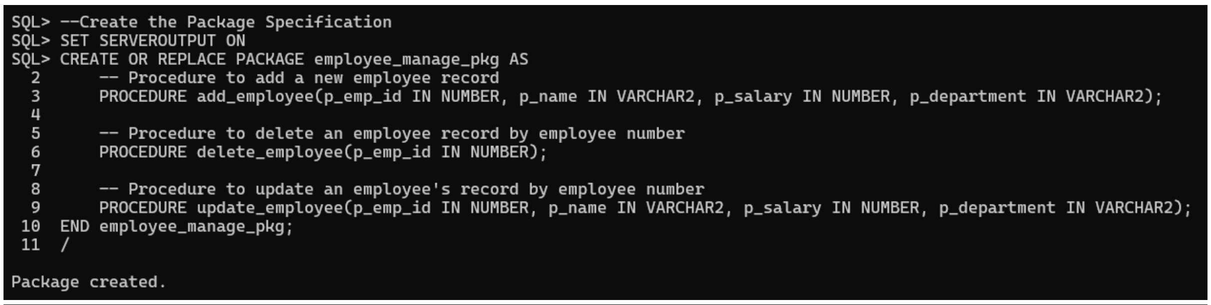
- Insert record and display total number records



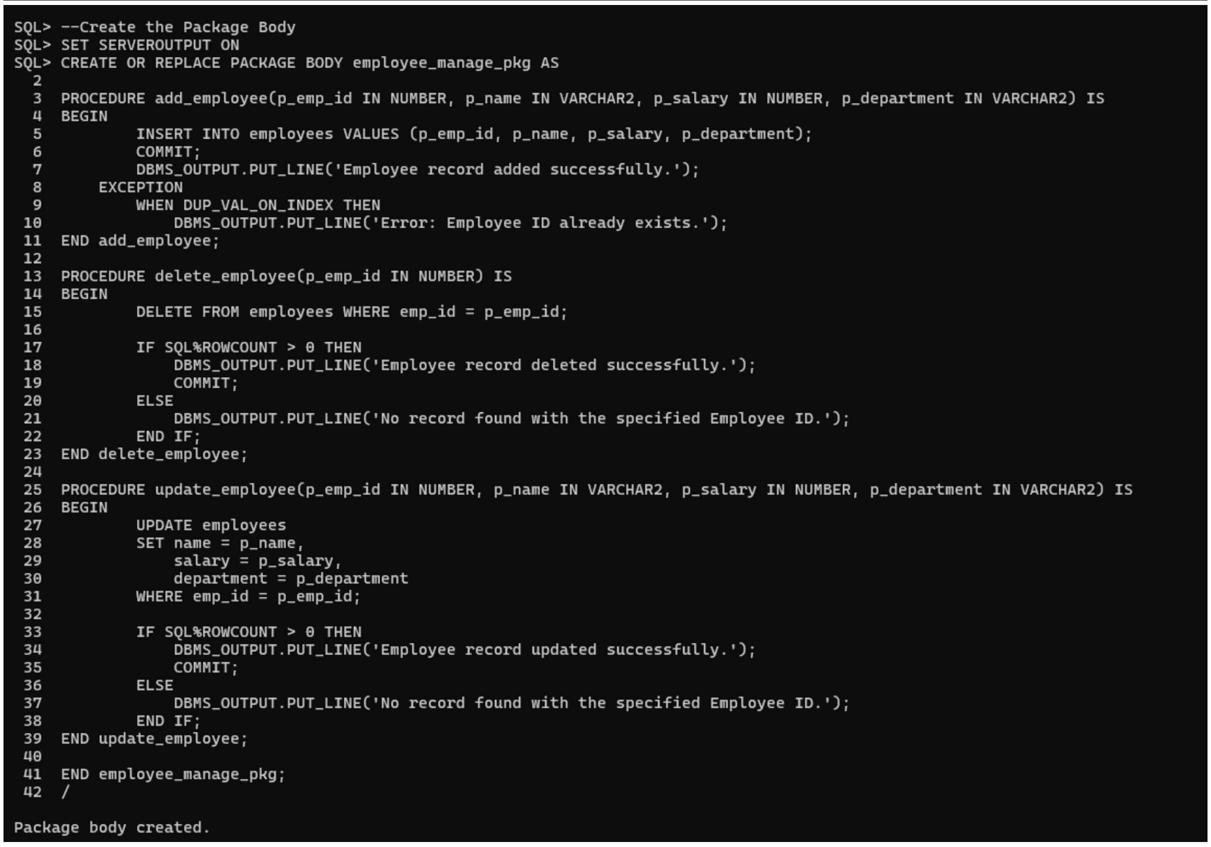
```
SQL>
SQL> --Using the Package
SQL> BEGIN
         employee_pkg.insert_employee(3, 'vijay', 45000, 'IT');
     END;
     /
Employee record inserted successfully.

PL/SQL procedure successfully completed.

SQL> --the Total Record Count
SQL> DECLARE
         v_total_records NUMBER;
     BEGIN
         v_total_records := employee_pkg.get_total_records;
         DBMS_OUTPUT.PUT_LINE('Total Number of Records: ' || v_total_records);
     END;
     /
Total Number of Records: 3

PL/SQL procedure successfully completed.
```

2. Create a package that will use 3 procedures as to add a new record as per user input, to delete a record as per given Employee no and update record as per modified data entered by the user for Emp_No.

- Write package specification and package body

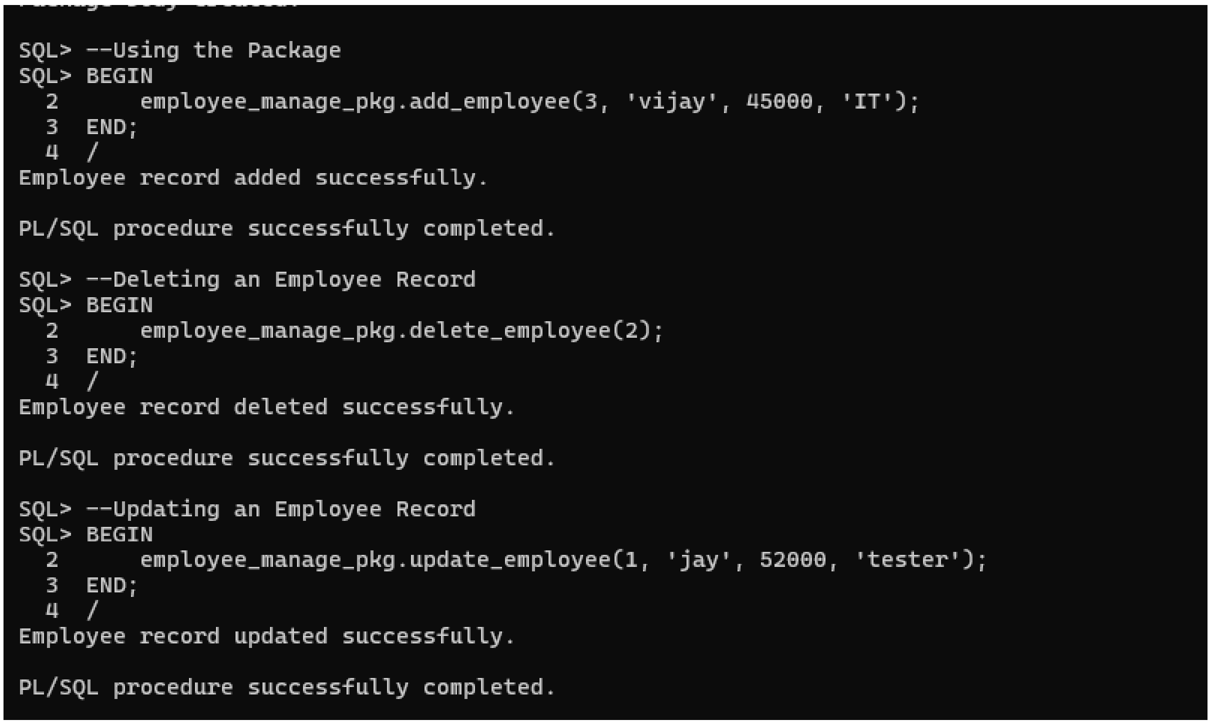


```
SQL> --Create the Package Specification
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PACKAGE employee_manage_pkg AS
         -- Procedure to add a new employee record
         PROCEDURE add_employee(p_emp_id IN NUMBER, p_name IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2);

         -- Procedure to delete an employee record by employee number
         PROCEDURE delete_employee(p_emp_id IN NUMBER);

         -- Procedure to update an employee's record by employee number
         PROCEDURE update_employee(p_emp_id IN NUMBER, p_name IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2);
     END employee_manage_pkg;
     /

Package created.
```



```
SQL> --Create the Package Body
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PACKAGE BODY employee_manage_pkg AS

     PROCEDURE add_employee(p_emp_id IN NUMBER, p_name IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2) IS
     BEGIN
             INSERT INTO employees VALUES (p_emp_id, p_name, p_salary, p_department);
             COMMIT;
             DBMS_OUTPUT.PUT_LINE('Employee record added successfully.');
         EXCEPTION
             WHEN DUP_VAL_ON_INDEX THEN
                 DBMS_OUTPUT.PUT_LINE('Error: Employee ID already exists.');
     END add_employee;

     PROCEDURE delete_employee(p_emp_id IN NUMBER) IS
     BEGIN
             DELETE FROM employees WHERE emp_id = p_emp_id;

             IF SQL%ROWCOUNT > 0 THEN
                 DBMS_OUTPUT.PUT_LINE('Employee record deleted successfully.');
                 COMMIT;
             ELSE
                 DBMS_OUTPUT.PUT_LINE('No record found with the specified Employee ID.');
             END IF;
     END delete_employee;

     PROCEDURE update_employee(p_emp_id IN NUMBER, p_name IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2) IS
     BEGIN
             UPDATE employees
             SET name = p_name,
                 salary = p_salary,
                 department = p_department
             WHERE emp_id = p_emp_id;

             IF SQL%ROWCOUNT > 0 THEN
                 DBMS_OUTPUT.PUT_LINE('Employee record updated successfully.');
                 COMMIT;
             ELSE
                 DBMS_OUTPUT.PUT_LINE('No record found with the specified Employee ID.');
             END IF;
     END update_employee;

     END employee_manage_pkg;
     /

Package body created.
```

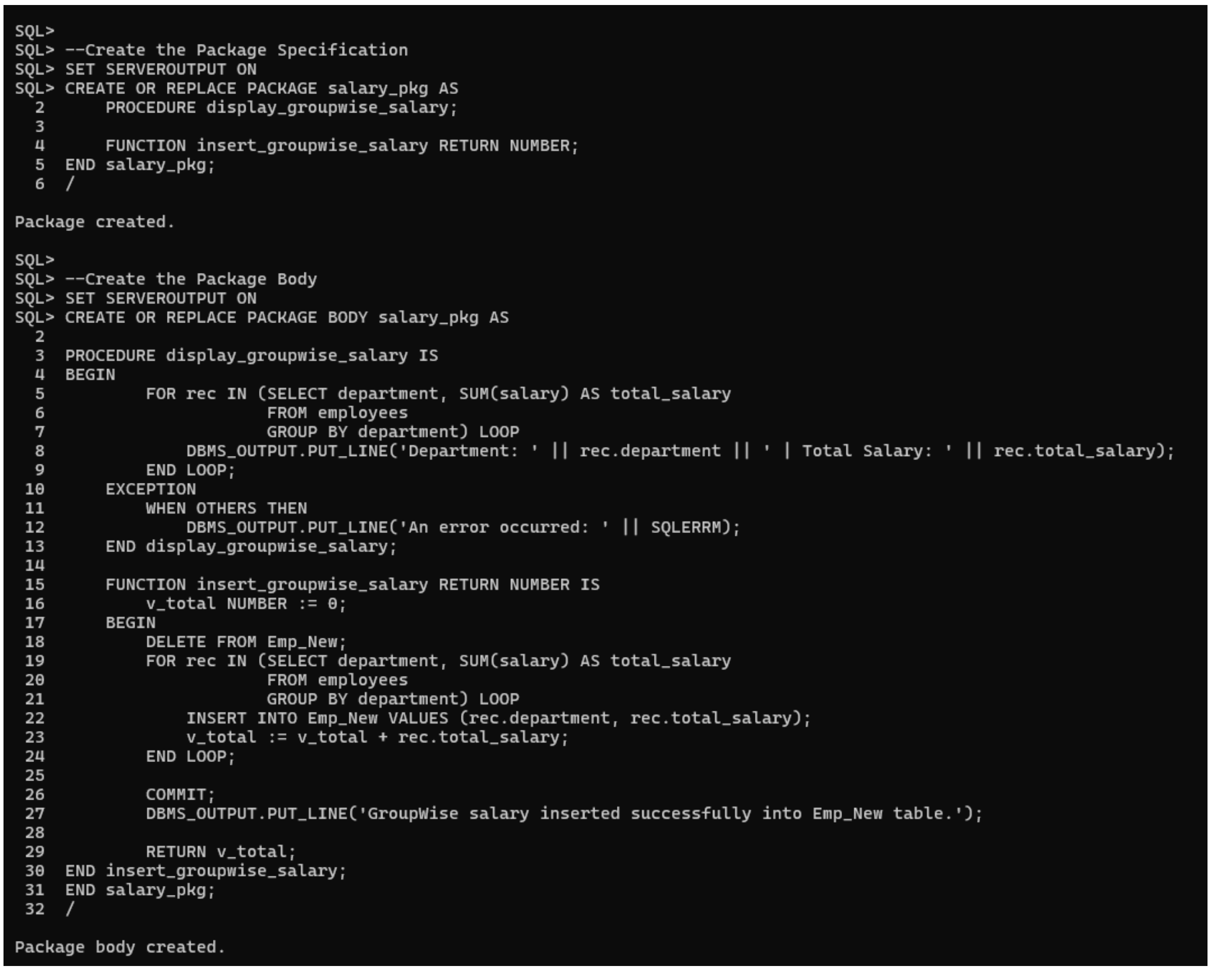
- Using procedures to insert,delete,update record



```
SQL> --Using the Package
SQL> BEGIN
  2      employee_manage_pkg.add_employee(3, 'vijay', 45000, 'IT');
  3  END;
  4  /
Employee record added successfully.

PL/SQL procedure successfully completed.

SQL> --Deleting an Employee Record
SQL> BEGIN
  2      employee_manage_pkg.delete_employee(2);
  3  END;
  4  /
Employee record deleted successfully.

PL/SQL procedure successfully completed.

SQL> --Updating an Employee Record
SQL> BEGIN
  2      employee_manage_pkg.update_employee(1, 'jay', 52000, 'tester');
  3  END;
  4  /
Employee record updated successfully.

PL/SQL procedure successfully completed.
```

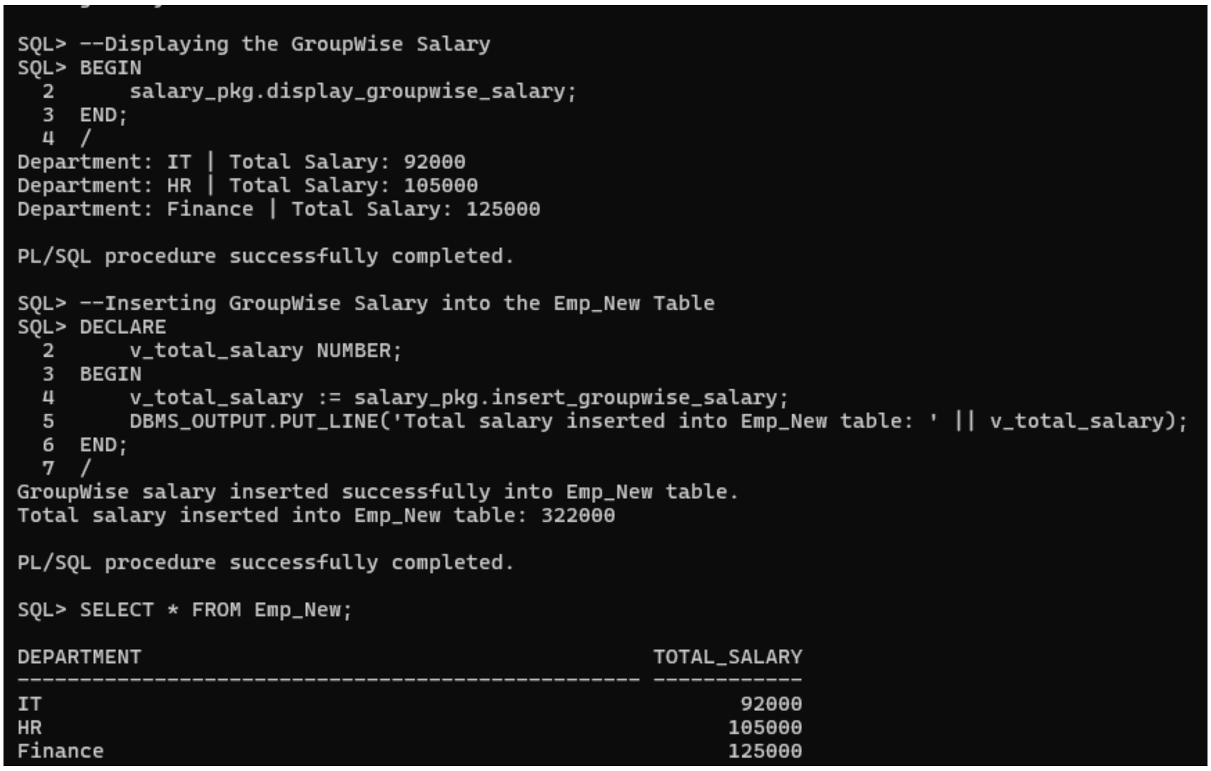
3. Create a package that will use procedure to display GroupWise salary and function that inserts GroupWise salary into Emp_New table and display appropriate messages.

- Write package specification and package body



```
SQL>
SQL> --Create the Package Specification
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PACKAGE salary_pkg AS
  2      PROCEDURE display_groupwise_salary;
  3
  4      FUNCTION insert_groupwise_salary RETURN NUMBER;
  5  END salary_pkg;
  6  /

Package created.

SQL>
SQL> --Create the Package Body
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PACKAGE BODY salary_pkg AS
  2
  3  PROCEDURE display_groupwise_salary IS
  4  BEGIN
  5          FOR rec IN (SELECT department, SUM(salary) AS total_salary
  6                      FROM employees
  7                      GROUP BY department) LOOP
  8              DBMS_OUTPUT.PUT_LINE('Department: ' || rec.department || ' | Total Salary: ' || rec.total_salary);
  9          END LOOP;
 10      EXCEPTION
 11          WHEN OTHERS THEN
 12              DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
 13      END display_groupwise_salary;
 14
 15      FUNCTION insert_groupwise_salary RETURN NUMBER IS
 16          v_total NUMBER := 0;
 17      BEGIN
 18          DELETE FROM Emp_New;
 19          FOR rec IN (SELECT department, SUM(salary) AS total_salary
 20                      FROM employees
 21                      GROUP BY department) LOOP
 22              INSERT INTO Emp_New VALUES (rec.department, rec.total_salary);
 23              v_total := v_total + rec.total_salary;
 24          END LOOP;
 25
 26          COMMIT;
 27          DBMS_OUTPUT.PUT_LINE('GroupWise salary inserted successfully into Emp_New table.');
 28
 29          RETURN v_total;
 30  END insert_groupwise_salary;
 31  END salary_pkg;
 32  /

Package body created.
```

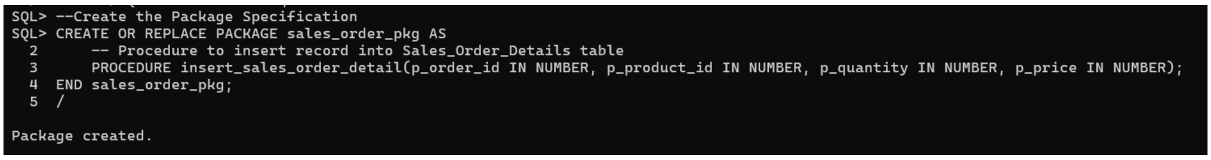
- Display groupwise salary and insert groupwise salary in emp_new table



```
SQL> --Displaying the GroupWise Salary
SQL> BEGIN
         salary_pkg.display_groupwise_salary;
     END;
     /
Department: IT | Total Salary: 92000
Department: HR | Total Salary: 105000
Department: Finance | Total Salary: 125000

PL/SQL procedure successfully completed.

SQL> --Inserting GroupWise Salary into the Emp_New Table
SQL> DECLARE
         v_total_salary NUMBER;
     BEGIN
         v_total_salary := salary_pkg.insert_groupwise_salary;
         DBMS_OUTPUT.PUT_LINE('Total salary inserted into Emp_New table: ' || v_total_salary);
     END;
     /
GroupWise salary inserted successfully into Emp_New table.
Total salary inserted into Emp_New table: 322000

PL/SQL procedure successfully completed.

SQL> SELECT * FROM Emp_New;

DEPARTMENT                                          TOTAL_SALARY
--------------------------------------------------- ------------
IT                                                         92000
HR                                                        105000
Finance                                                   125000
```

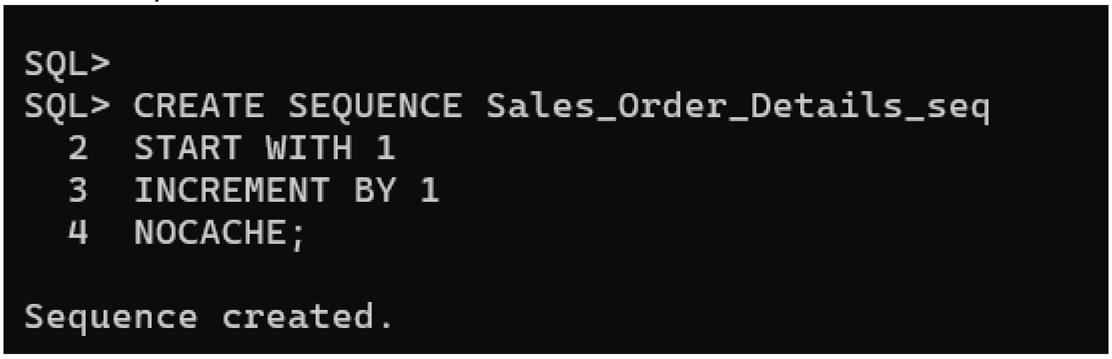
4. Create a package that will insert a record in Sales_Order_Details table on the base of Customer_Master, Sales_Order, Salesman_Master table, if the status of the order is fulfilled

- Write package specification



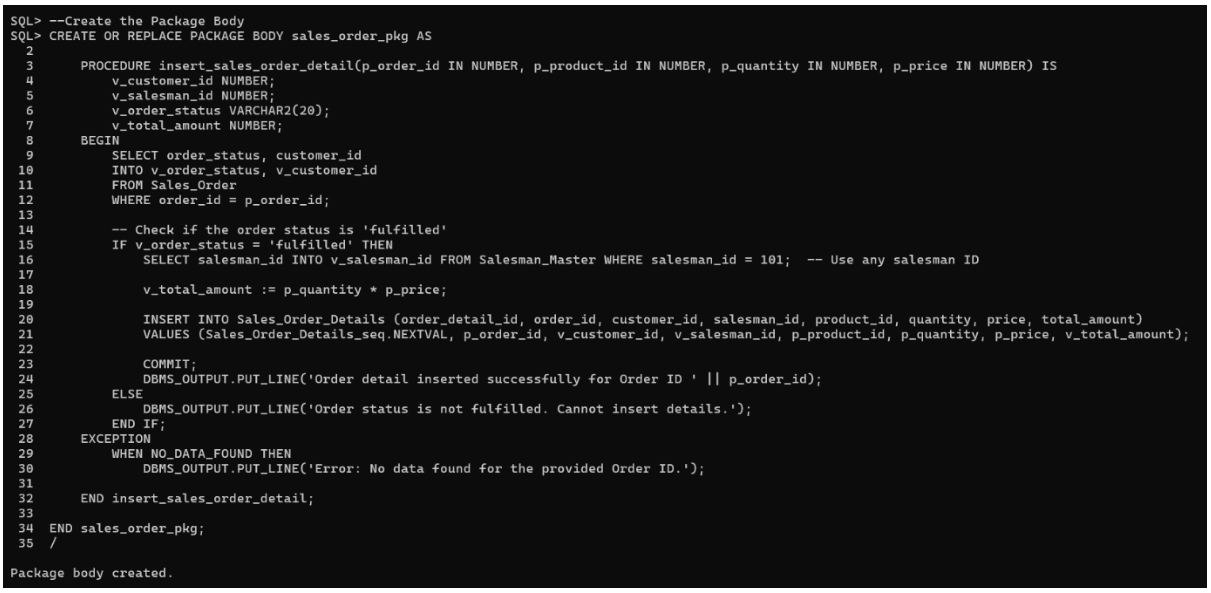
```
SQL> --Create the Package Specification
SQL> CREATE OR REPLACE PACKAGE sales_order_pkg AS
         -- Procedure to insert record into Sales_Order_Details table
         PROCEDURE insert_sales_order_detail(p_order_id IN NUMBER, p_product_id IN NUMBER, p_quantity IN NUMBER, p_price IN NUMBER);
     END sales_order_pkg;
     /

Package created.
```

- Create sequence



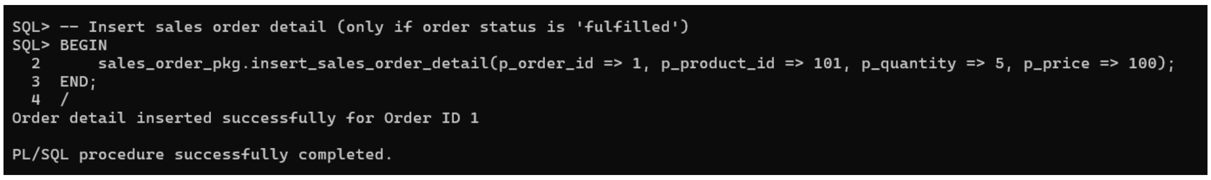
```
SQL>
SQL> CREATE SEQUENCE Sales_Order_Details_seq
     START WITH 1
     INCREMENT BY 1
     NOCACHE;

Sequence created.
```

- Create package body



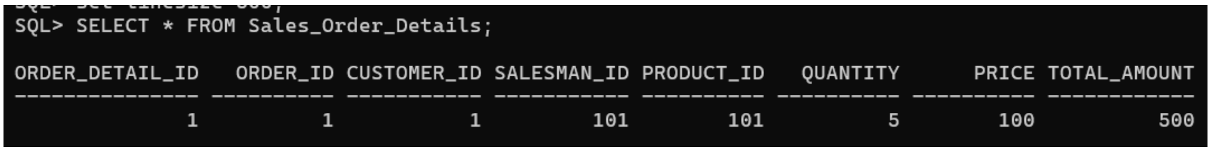
```
SQL> --Create the Package Body
SQL> CREATE OR REPLACE PACKAGE BODY sales_order_pkg AS

    PROCEDURE insert_sales_order_detail(p_order_id IN NUMBER, p_product_id IN NUMBER, p_quantity IN NUMBER, p_price IN NUMBER) IS
        v_customer_id NUMBER;
        v_salesman_id NUMBER;
        v_order_status VARCHAR2(20);
        v_total_amount NUMBER;
    BEGIN
        SELECT order_status, customer_id
        INTO v_order_status, v_customer_id
        FROM Sales_Order
        WHERE order_id = p_order_id;

        -- Check if the order status is 'fulfilled'
        IF v_order_status = 'fulfilled' THEN
            SELECT salesman_id INTO v_salesman_id FROM Salesman_Master WHERE salesman_id = 101;  -- Use any salesman ID

            v_total_amount := p_quantity * p_price;

            INSERT INTO Sales_Order_Details (order_detail_id, order_id, customer_id, salesman_id, product_id, quantity, price, total_amount)
            VALUES (Sales_Order_Details_seq.NEXTVAL, p_order_id, v_customer_id, v_salesman_id, p_product_id, p_quantity, p_price, v_total_amount);

            COMMIT;
            DBMS_OUTPUT.PUT_LINE('Order detail inserted successfully for Order ID ' || p_order_id);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Order status is not fulfilled. Cannot insert details.');
        END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Error: No data found for the provided Order ID.');

    END insert_sales_order_detail;

END sales_order_pkg;
/

Package body created.
```

- Insert sales order details (only if order status is fulfilled)



```
SQL> -- Insert sales order detail (only if order status is 'fulfilled')
SQL> BEGIN
    sales_order_pkg.insert_sales_order_detail(p_order_id => 1, p_product_id => 101, p_quantity => 5, p_price => 100);
END;
/
Order detail inserted successfully for Order ID 1

PL/SQL procedure successfully completed.
```

- Display records



```
SQL> SELECT * FROM Sales_Order_Details;

ORDER_DETAIL_ID   ORDER_ID CUSTOMER_ID SALESMAN_ID PRODUCT_ID   QUANTITY      PRICE TOTAL_AMOUNT
--------------- ---------- ----------- ----------- ---------- ---------- ---------- ------------
              1          1           1         101        101          5        100          500
```