

Advanced Database - TP 2 - Part 2

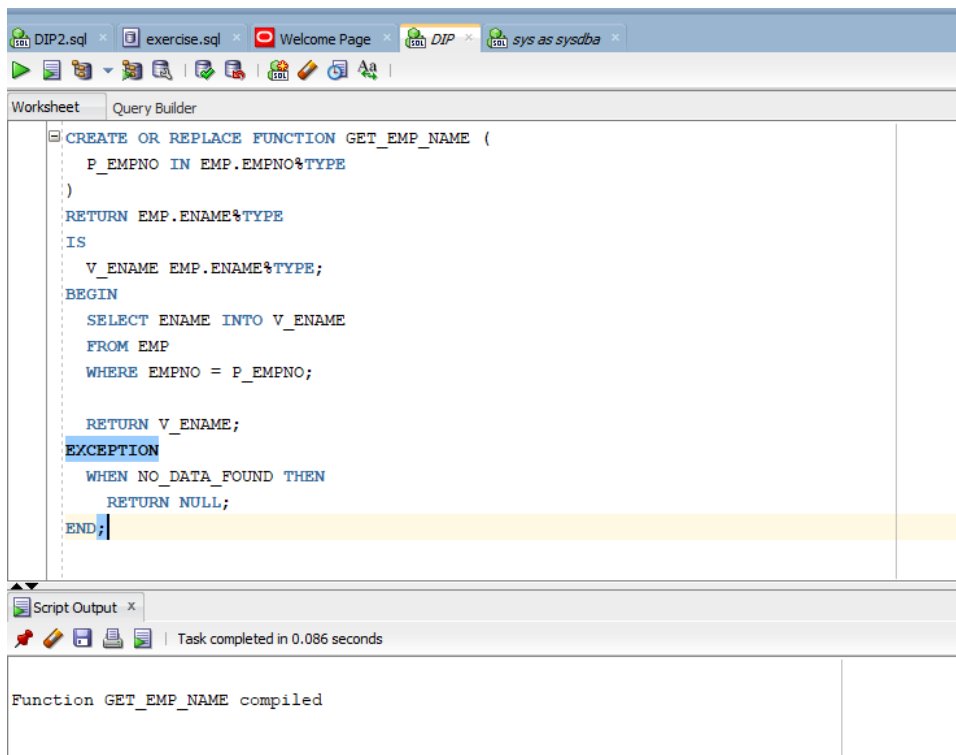
PL/SQL

Name: Sayed Mujtaba

Lastname: Ahmady

STDNO: 62707

1. Create a function that gets an employee's name from it's empno.



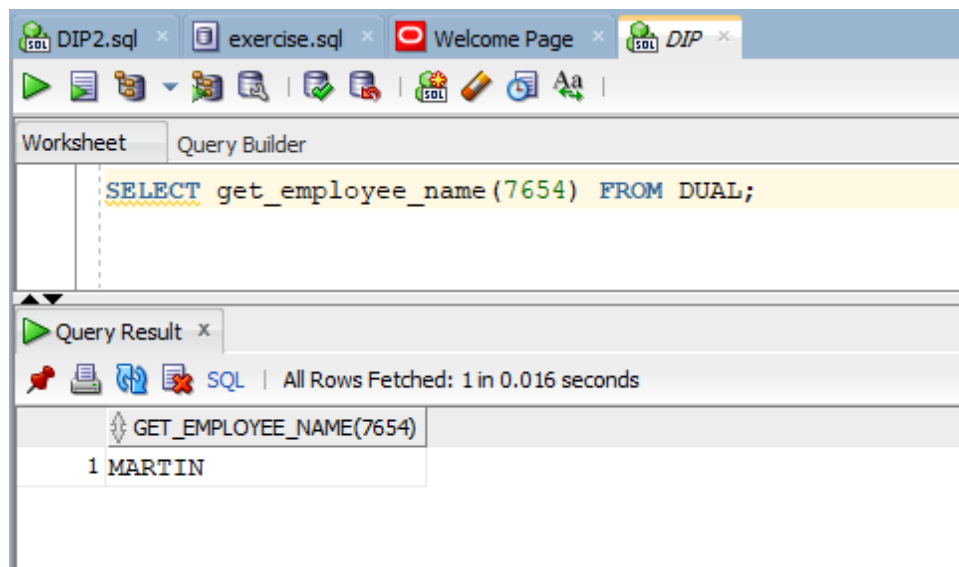
The screenshot displays the Oracle SQL Developer environment. The top toolbar includes icons for running, saving, and other database operations. The main window is titled 'Worksheet' and contains the following PL/SQL code:

```
CREATE OR REPLACE FUNCTION GET_EMP_NAME (  
    P_EMPNO IN EMP.EMPNO%TYPE  
)  
RETURN EMP.ENAME%TYPE  
IS  
    V_ENAME EMP.ENAME%TYPE;  
BEGIN  
    SELECT ENAME INTO V_ENAME  
    FROM EMP  
    WHERE EMPNO = P_EMPNO;  
  
    RETURN V_ENAME;  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        RETURN NULL;  
END;
```

Below the code editor, the 'Script Output' window shows the message: 'Function GET_EMP_NAME compiled'. Above this message, it states 'Task completed in 0.086 seconds'.

2. Test the query with

SELECT FunctionName(7654) FROM DUAL;



Exercise 2. Procedure & Display & Cursors

The screenshot shows the Oracle SQL Developer interface. The top window is titled 'exercise.sql' and contains the following PL/SQL code:

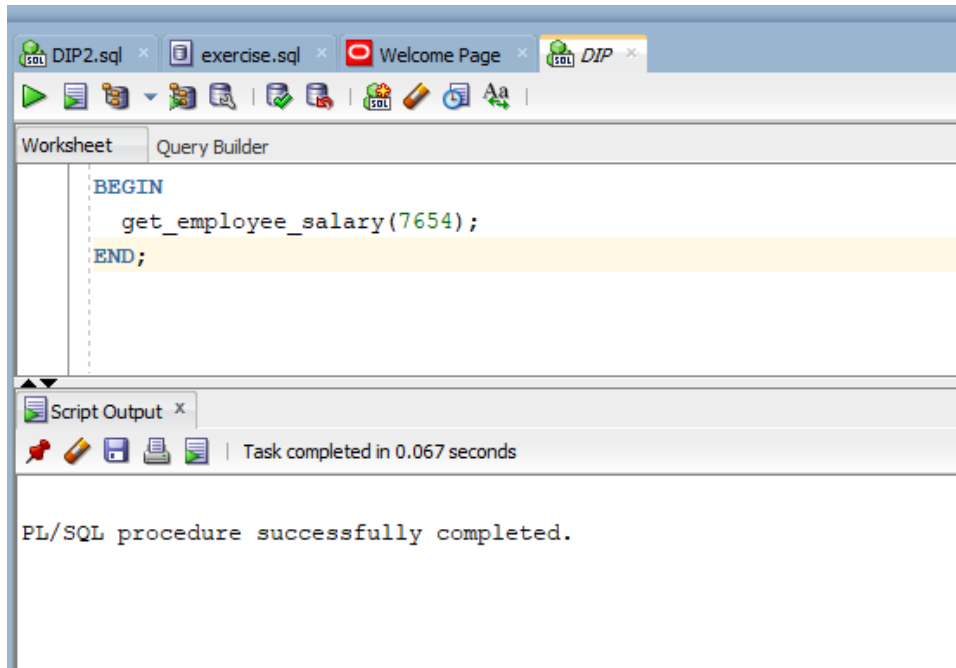
```
CREATE OR REPLACE PROCEDURE get_employee_salary
(p_empno IN EMP.EMPNO%TYPE)
IS
v_net_salary NUMBER(7, 2);
v_avg_salary NUMBER(7, 2);
BEGIN
SELECT (SAL + COALESCE(COMM, 0)) * 0.8 -- assumes 20% tax rate
INTO v_net_salary
FROM EMP
WHERE EMPNO = p_empno;

SELECT AVG(SAL + COALESCE(COMM, 0)) * 0.8 -- assumes 20% tax rate
INTO v_avg_salary
FROM EMP
WHERE JOB = (SELECT JOB FROM EMP WHERE EMPNO = p_empno);

DBMS_OUTPUT.PUT_LINE('Net salary: ' || v_net_salary);
DBMS_OUTPUT.PUT_LINE('Average salary for job: ' || v_avg_salary);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Employee not found.');
```

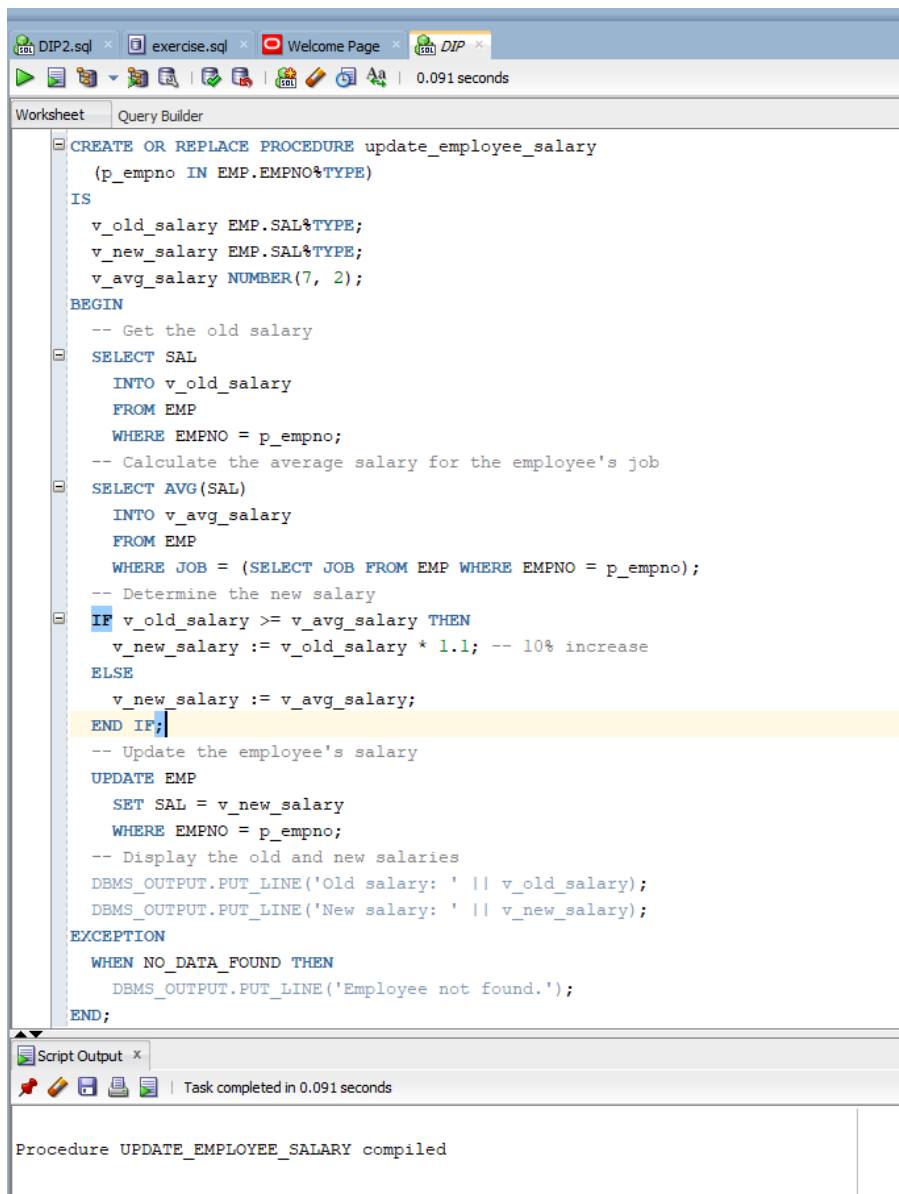
The bottom window is titled 'Script Output' and shows the message: 'Task completed in 0.091 seconds' and 'Procedure GET_EMPLOYEE_SALARY compiled'.

This will display the net salary and average salary for the employee with empno 7654 (assuming that employee exists in the EMP table). The DBMS_OUTPUT.PUT_LINE statements are used to display the results in the console.



Exercise 3. Procedure & Update

This procedure takes an employee number (empno) as input and updates the employee's salary based on whether their current salary is greater than or equal to the average salary for their job. If their salary is higher than or equal to the average, it increases their salary by 10%. If it's lower than the average, their new salary becomes the average.



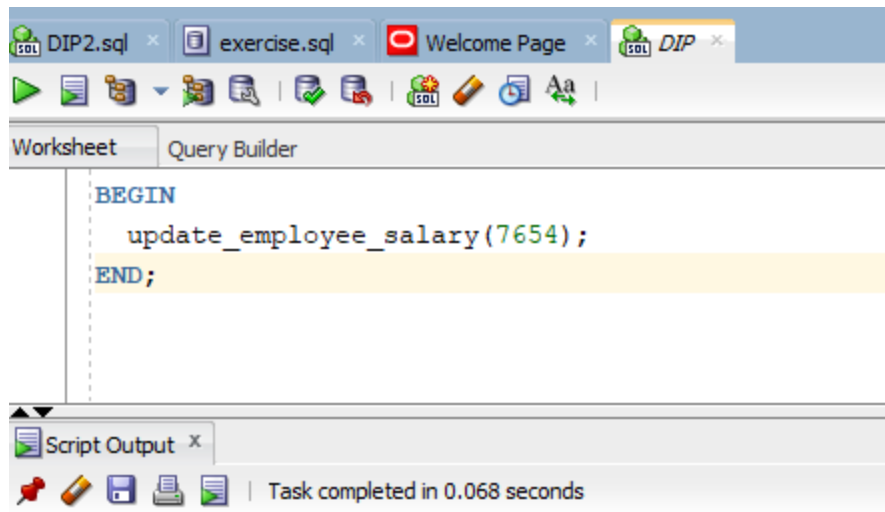
The screenshot shows a SQL IDE window with a tab titled 'exercise.sql'. The main editor displays a PL/SQL procedure named 'update_employee_salary'. The procedure takes a parameter 'p_empno' of type 'EMP.EMPNO%TYPE'. It declares three variables: 'v_old_salary' of type 'EMP.SAL%TYPE', 'v_new_salary' of type 'EMP.SAL%TYPE', and 'v_avg_salary' of type 'NUMBER(7, 2)'. The procedure begins by getting the old salary for the employee with 'p_empno' and calculating the average salary for their job. It then uses an 'IF' statement to determine the new salary: if the old salary is greater than or equal to the average salary, it increases the old salary by 10%; otherwise, it sets the new salary to the average salary. Finally, it updates the employee's salary and displays both the old and new salaries using 'DBMS_OUTPUT.PUT_LINE'. An exception block handles the 'NO_DATA_FOUND' error by displaying a message. The procedure ends with 'END;'. Below the editor, the 'Script Output' pane shows the message 'Task completed in 0.091 seconds' and 'Procedure UPDATE_EMPLOYEE_SALARY compiled'.

```
CREATE OR REPLACE PROCEDURE update_employee_salary
(p_empno IN EMP.EMPNO%TYPE)
IS
    v_old_salary EMP.SAL%TYPE;
    v_new_salary EMP.SAL%TYPE;
    v_avg_salary NUMBER(7, 2);
BEGIN
    -- Get the old salary
    SELECT SAL
    INTO v_old_salary
    FROM EMP
    WHERE EMPNO = p_empno;
    -- Calculate the average salary for the employee's job
    SELECT AVG(SAL)
    INTO v_avg_salary
    FROM EMP
    WHERE JOB = (SELECT JOB FROM EMP WHERE EMPNO = p_empno);
    -- Determine the new salary
    IF v_old_salary >= v_avg_salary THEN
        v_new_salary := v_old_salary * 1.1; -- 10% increase
    ELSE
        v_new_salary := v_avg_salary;
    END IF;
    -- Update the employee's salary
    UPDATE EMP
    SET SAL = v_new_salary
    WHERE EMPNO = p_empno;
    -- Display the old and new salaries
    DBMS_OUTPUT.PUT_LINE('Old salary: ' || v_old_salary);
    DBMS_OUTPUT.PUT_LINE('New salary: ' || v_new_salary);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee not found.');
```

Task completed in 0.091 seconds

Procedure UPDATE_EMPLOYEE_SALARY compiled

This will update the salary for the employee with empno 7654 and display their old and new salaries.



PL/SQL procedure successfully completed.

Exercise 4. Procedure

This procedure loops through the EMP table and calculates bonuses for employees based on their job. For salesmen, it doubles their commission ($COMM * 2$). For clerks, it increases their salary by 15% ($SAL * 1.15$). For managers, it increases their salary by 18% ($SAL * 1.18$). The calculated bonuses are then inserted into the BONUS table.

```
CREATE OR REPLACE PROCEDURE calculate_bonuses
IS
BEGIN
    -- Calculate bonuses for salesmen
    FOR s IN (SELECT * FROM EMP WHERE JOB = 'SALESMAN')
    LOOP
        INSERT INTO BONUS (ENAME, JOB, SAL, COMM)
        VALUES (s.ENAME, s.JOB, s.SAL, s.COMM * 2);
    END LOOP;

    -- Calculate bonuses for clerks
    FOR c IN (SELECT * FROM EMP WHERE JOB = 'CLERK')
    LOOP
        INSERT INTO BONUS (ENAME, JOB, SAL, COMM)
        VALUES (c.ENAME, c.JOB, c.SAL * 1.15, c.COMM);
    END LOOP;

    -- Calculate bonuses for managers
    FOR m IN (SELECT * FROM EMP WHERE JOB = 'MANAGER')
    LOOP
        INSERT INTO BONUS (ENAME, JOB, SAL, COMM)
        VALUES (m.ENAME, m.JOB, m.SAL * 1.18, m.COMM);
    END LOOP;
END;
```

Script Output x

Task completed in 0.263 seconds

Procedure CALCULATE_BONUSES compiled

This will calculate and insert bonuses for all employees in the EMP table.

```
BEGIN
    calculate_bonuses;
END;
```

Script Output x

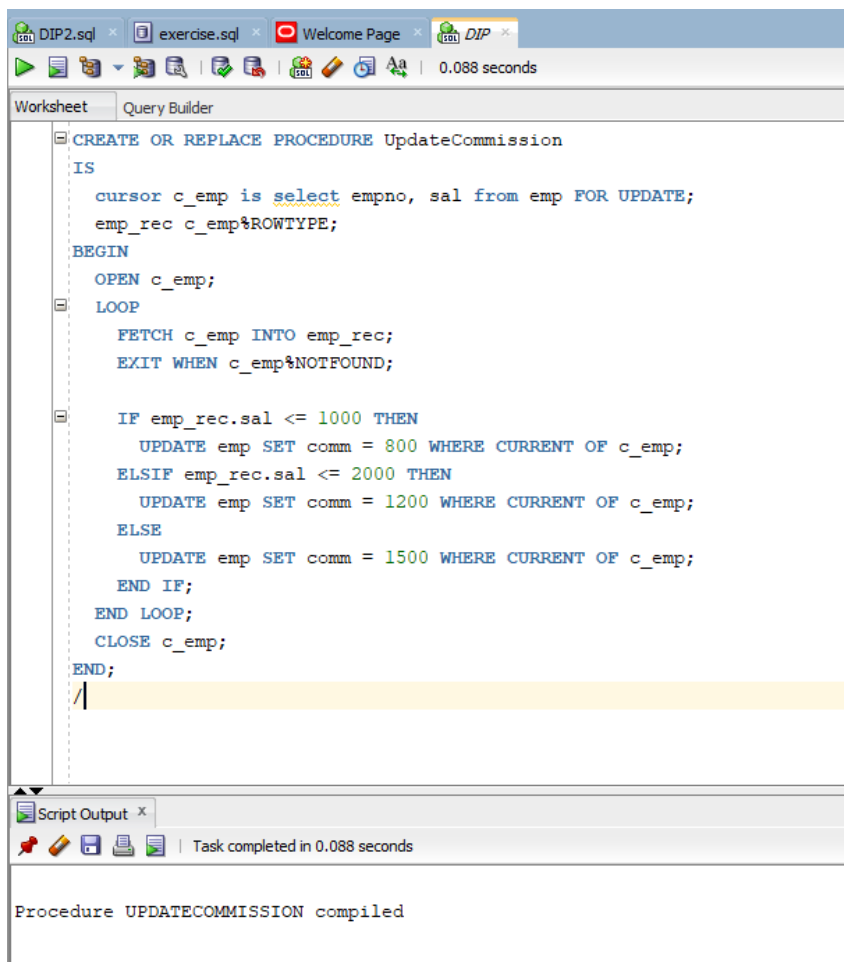
Task completed in 0.07 seconds

PL/SQL procedure successfully completed.

Exercise 5. SELECT for UPDATE

This procedure first opens a cursor that selects the employee's empno and sal columns from the emp table with the FOR UPDATE clause. This puts a lock on the selected rows, ensuring that other transactions cannot modify them at the same time.

Then, it loops through the cursor and checks the employee's salary. Based on their salary, it updates their commission using the UPDATE statement and the WHERE CURRENT OF cursor clause to update the current row being processed by the cursor.



```
CREATE OR REPLACE PROCEDURE UpdateCommission
IS
  cursor c_emp is select empno, sal from emp FOR UPDATE;
  emp_rec c_emp%ROWTYPE;
BEGIN
  OPEN c_emp;
  LOOP
    FETCH c_emp INTO emp_rec;
    EXIT WHEN c_emp%NOTFOUND;

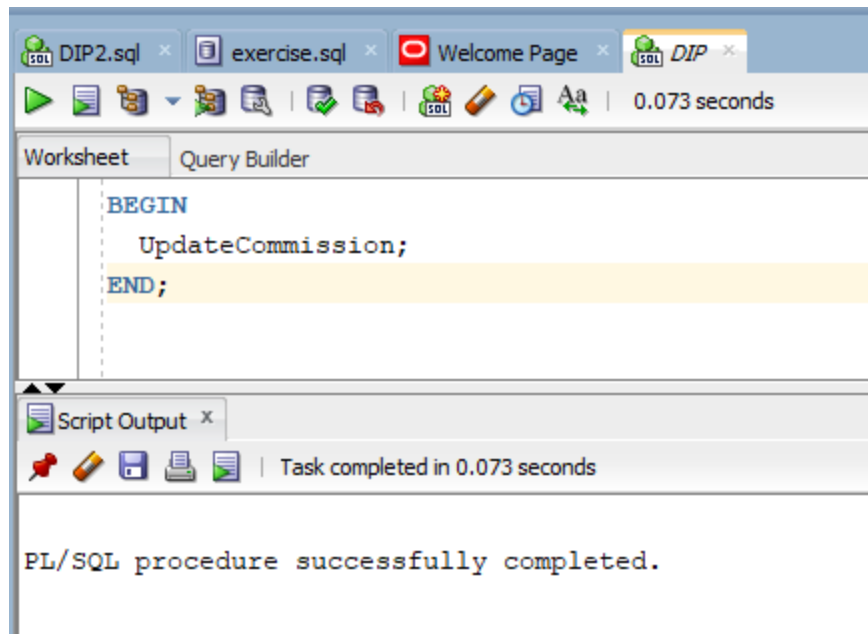
    IF emp_rec.sal <= 1000 THEN
      UPDATE emp SET comm = 800 WHERE CURRENT OF c_emp;
    ELSIF emp_rec.sal <= 2000 THEN
      UPDATE emp SET comm = 1200 WHERE CURRENT OF c_emp;
    ELSE
      UPDATE emp SET comm = 1500 WHERE CURRENT OF c_emp;
    END IF;
  END LOOP;
  CLOSE c_emp;
END;
```

Script Output x

Task completed in 0.088 seconds

Procedure UPDATECOMMISSION compiled

This will update the commission of all employees in the EMP table based on their salary.



Exercise 6. Condition on EXIT loop

This procedure first opens a cursor that selects the employee's empno and sal columns from the emp table and orders them by salary in descending order.

Then, it loops through the cursor using FETCH to retrieve each row. The loop also checks two conditions: whether the end of the cursor has been reached using the NOTFOUND attribute of the cursor, and whether the loop has already processed the 5 highest and 5 lowest salaries using a counter variable i.

Inside the loop, the procedure checks whether the current employee is one of the 5 highest or 5 lowest salaries, and displays their empno and sal accordingly using the dbms_output.put_line procedure.

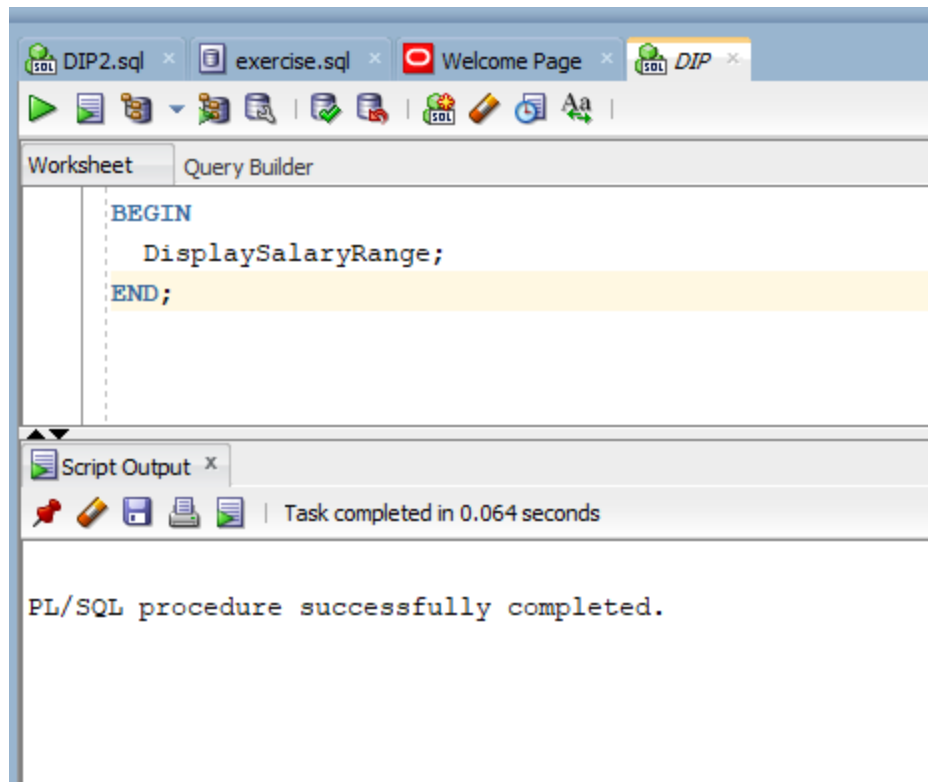
```
CREATE OR REPLACE PROCEDURE DisplaySalaryRange
IS
    cursor c_emp is select empno, sal from emp order by sal desc;
    emp_rec c_emp%ROWTYPE;
    i number := 0;
BEGIN
    OPEN c_emp;
    LOOP
        FETCH c_emp INTO emp_rec;
        EXIT WHEN c_emp%NOTFOUND OR i >= 10;
        i := i + 1;
        IF i <= 5 THEN
            dbms_output.put_line(emp_rec.empno || ' - ' || emp_rec.sal || ' (highest)');
        ELSE
            dbms_output.put_line(emp_rec.empno || ' - ' || emp_rec.sal || ' (lowest)');
        END IF;
    END LOOP;
    CLOSE c_emp;
END;
```

Script Output x

Task completed in 0.069 seconds

Procedure DISPLAYSALARYRANGE compiled

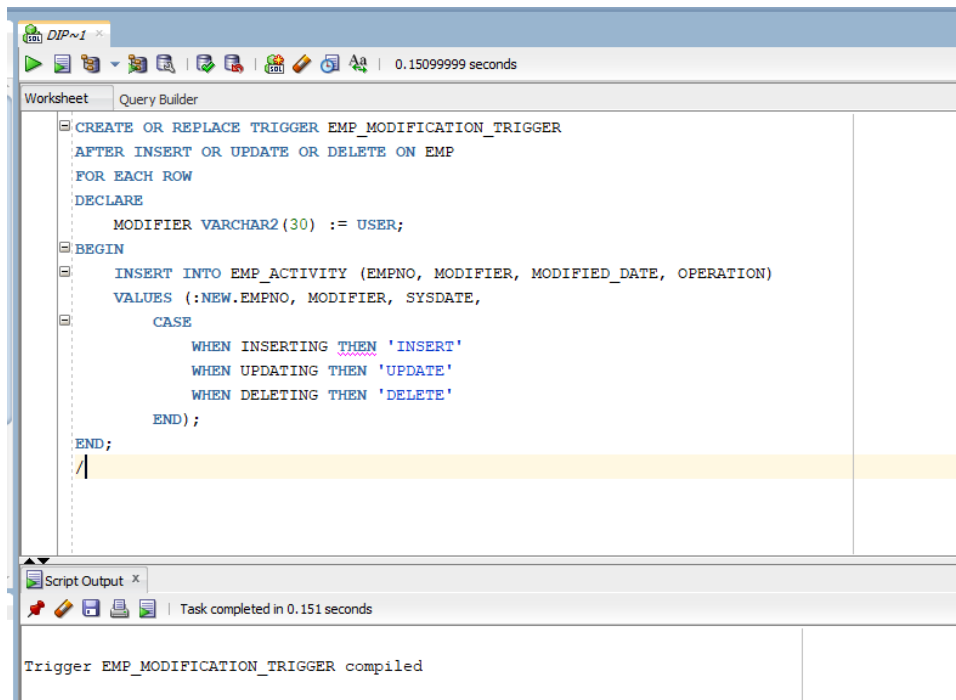
This will display the empno and sal of the 5 employees with the highest salaries, followed by the 5 employees with the lowest salaries.



Exercise 7. Triggers

1. Create The triggers:

This trigger will insert a new row in the emp_modification_history table every time an insert, update, or delete operation is performed on the emp table. The row will contain the name of the user who performed the operation, the date and time of the operation, and the type of operation.



2. Function to analyze results:

This function takes two optional parameters: `p_user_name` and `p_date`. If both parameters are null, it will return the total number of modifications performed on the `emp` table since it was created. If `p_user_name` is not null, it will return the total number of modifications performed by that user. If `p_date` is not null, it will return the total number of modifications performed on that date.

The function uses the `emp_modification_history` table to count the number of modifications that match the specified criteria. It also uses the `USER_OBJECTS` view to get the creation date of the `emp_modification_history` table, so that it can filter out modifications that were performed before the table was created.

Worksheet Query Builder

```
CREATE OR REPLACE FUNCTION analyze_activity(p_user_name IN VARCHAR2 DEFAULT NULL, p_date IN DATE DEFAULT NULL)
RETURN NUMBER
IS
    v_count NUMBER := 0;
BEGIN
    SELECT COUNT(*)
    INTO v_count
    FROM emp_modification_history
    WHERE (user_name = p_user_name OR p_user_name IS NULL)
    AND (TRUNC(modification_date) = TRUNC(p_date) OR p_date IS NULL)
    AND (modification_date >= (SELECT MIN(CREATED)
                                FROM USER_OBJECTS
                                WHERE OBJECT_NAME = 'EMP_MODIFICATION_HISTORY'));

    RETURN v_count;
END;
```

Script Output x

Task completed in 0.078 seconds

Function ANALYZE_ACTIVITY compiled