

# 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 shows the Oracle SQL Developer interface. In the top navigation bar, there are tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', 'DIP', and 'sys as sysdba'. Below the tabs is a toolbar with various icons. The main area 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 worksheet is a 'Script Output' window which displays the message: "Function GET\_EMP\_NAME compiled".

2. Test the query with

```
SELECT FunctionName(7654) FROM DUAL;
```

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and 'DIP'. Below the menu is a toolbar with various icons for database operations. The main area has two tabs: 'Worksheet' and 'Query Builder', with 'Worksheet' selected. In the Worksheet tab, the following SQL code is entered:

```
SELECT get_employee_name(7654) FROM DUAL;
```

Below the worksheet is a 'Query Result' window. It has tabs for 'SQL' and 'Text', with 'SQL' selected. The status bar indicates 'All Rows Fetched: 1 in 0.016 seconds'. The result grid shows one row:

	GET_EMPLOYEE_NAME(7654)
1	MARTIN

## Exercice 2. Procedure & Display & Cursors

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and 'DIP'. Below the menu is a toolbar with various icons. The main workspace is titled 'Worksheet' and contains a 'Query Builder' tab. The code in the worksheet is as follows:

```
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.');
END;
/
```

Below the worksheet is a 'Script Output' window showing the result of the compilation:

```
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.

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and the active tab 'DIP'. Below the menu is a toolbar with various icons. The main workspace is titled 'Worksheet' and contains the following PL/SQL code:

```
BEGIN
    get_employee_salary(7654);
END;
```

The 'get\_employee\_salary' call is highlighted with a yellow background. Below the worksheet is a 'Script Output' window showing the result of the execution:

```
PL/SQL procedure successfully completed.
```

### Exercice 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 the Oracle SQL Developer interface with the 'Worksheet' tab selected. The code 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 local variables: 'v\_old\_salary', 'v\_new\_salary', and 'v\_avg\_salary'. The procedure begins by selecting the current salary for the employee with 'p\_empno' from the 'EMP' table. It then calculates the average salary for the employee's job. If the employee's current salary is greater than or equal to the average salary, the new salary is set to be 10% higher. Otherwise, it is set to the average salary. Finally, the procedure updates the employee's salary in the 'EMP' table and displays the old and new salaries using 'DBMS\_OUTPUT.PUT\_LINE'. An exception block handles the case where no data is found for the given employee number.

```
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.');
END;
```

Script Output | 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.

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and 'DIP'. Below the menu is a toolbar with various icons. The main workspace is titled 'Worksheet' and contains the following PL/SQL code:

```
BEGIN
    update_employee_salary(7654);
END;
```

The 'Script Output' window below shows the result of the execution:

```
PL/SQL procedure successfully completed.
```

## Exercice 4. Procedure

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

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and 'DIP'. The status bar at the bottom shows '0.26300001 seconds'. The main area is a 'Worksheet' tab where a PL/SQL procedure is being written:

```
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;
```

Below the worksheet is a 'Script Output' window showing the result of the compilation:

```
Procedure CALCULATE_BONUSES compiled
```

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

The screenshot shows the Oracle SQL Developer interface with the same layout as the previous one. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and 'DIP'. The status bar at the bottom shows '0.07 seconds'. The main area is a 'Worksheet' tab containing the following code:

```
BEGIN
    calculate_bonuses;
END;
```

Below the worksheet is a 'Script Output' window showing the result of the execution:

```
PL/SQL procedure successfully completed.
```

## Exercice 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.

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and 'DIP'. Below the menu is a toolbar with various icons. The main workspace is titled 'Worksheet' and contains the following PL/SQL code:

```
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;
```

The bottom pane is titled 'Script Output' and displays the message: 'Procedure UPDATECOMMISSION compiled'.

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

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and 'DIP'. Below the menu is a toolbar with various icons. The main workspace is titled 'Worksheet' and contains the following PL/SQL code:

```
BEGIN
    UpdateCommission;
END;
```

The 'Script Output' window below shows the result of the execution:

```
PL/SQL procedure successfully completed.
```

## Exercice 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.

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are three tabs: DIP2.sql, exercise.sql, and Welcome Page. The main window is titled "Worksheet" and contains the following PL/SQL code:

```
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;
```

In the bottom right corner of the worksheet area, there is a yellow status bar with the text "Task completed in 0.069 seconds".

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

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'DIP2.sql', 'exercise.sql', 'Welcome Page', and 'DIP'. Below the menu is a toolbar with various icons. The main workspace is titled 'Worksheet' and contains the following PL/SQL code:

```
BEGIN
    DisplaySalaryRange;
END;
```

Below the worksheet is a 'Script Output' window showing the result of the execution:

```
PL/SQL procedure successfully completed.
```

## Exercice 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.

The screenshot shows the Oracle SQL Developer interface. In the top window (Worksheet tab), a PL/SQL trigger creation script is displayed:

```
CREATE OR REPLACE TRIGGER EMP_MODIFICATION_TRIGGER
AFTER INSERT OR UPDATE OR DELETE ON EMP
FOR EACH ROW
DECLARE
    MODIFIER VARCHAR2(30) := USER;
BEGIN
    INSERT INTO EMP_ACTIVITY (EMPNO, MODIFIER, MODIFIED_DATE, OPERATION)
    VALUES (:NEW.EMPNO, MODIFIER, SYSDATE,
    CASE
        WHEN INSERTING THEN 'INSERT'
        WHEN UPDATING THEN 'UPDATE'
        WHEN DELETING THEN 'DELETE'
    END);
END;
```

In the bottom window (Script Output tab), the execution message is shown:

```
Trigger EMP_MODIFICATION_TRIGGER compiled
```

## 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.

DIP-1

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