

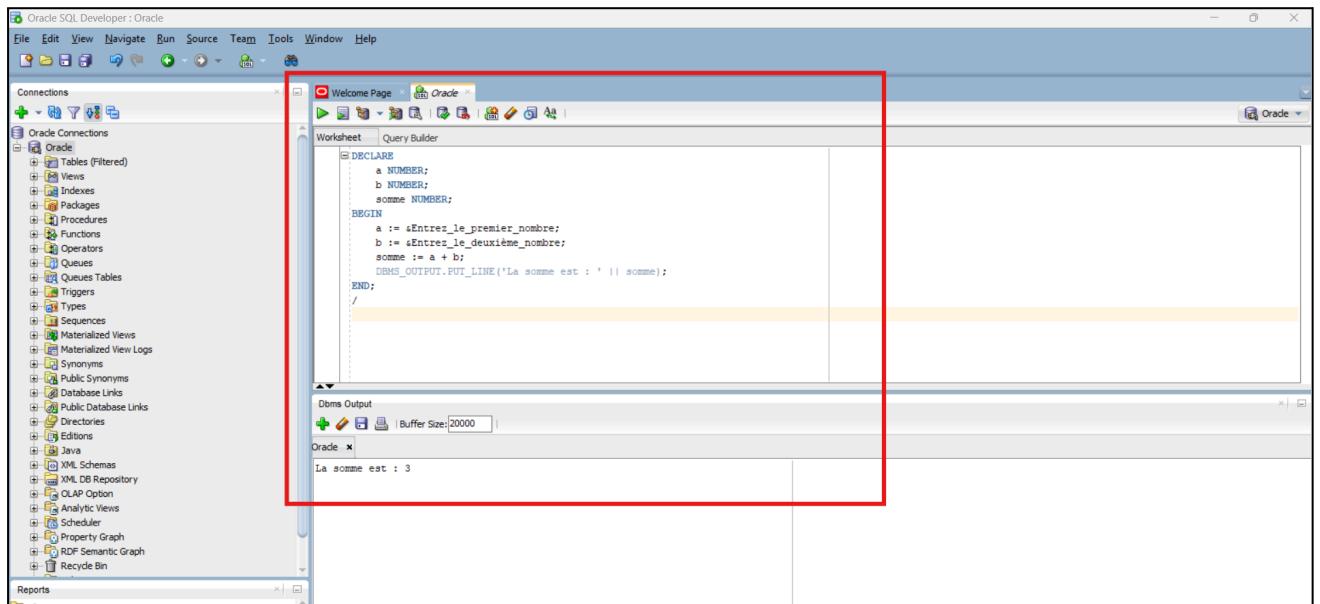
TP n° 4 : Introduction au Procedural Language (PL/SQL) d'Oracle

Téléchargement et installation des outils

- Téléchargement de SQL Developer (version 24.3.1) depuis le site officiel d'Oracle.
- Téléchargement et installation de Oracle Database Express Edition (XE) pour disposer d'une base de données locale sur la machine.
- Ouverture de SQL Developer après l'installation.

Exercice 1 : Programmation PL/SQL

1. Écrire une procédure anonyme PL/SQL qui permet de demander à un utilisateur de saisir deux entiers et d'afficher leur somme.



The screenshot shows the Oracle SQL Developer interface. On the left, the Connections sidebar shows a single connection named "Oracle". The main workspace has a red box highlighting the "Worksheet" tab. Inside the worksheet, there is a PL/SQL anonymous block:

```

DECLARE
    a NUMBER;
    b NUMBER;
    somme NUMBER;
BEGIN
    a := &Entrez_le_premier_nombre;
    b := &Entrez_le_deuxième_nombre;
    somme := a + b;
    DBMS_OUTPUT.PUT_LINE('La somme est : ' || somme);
END;
/

```

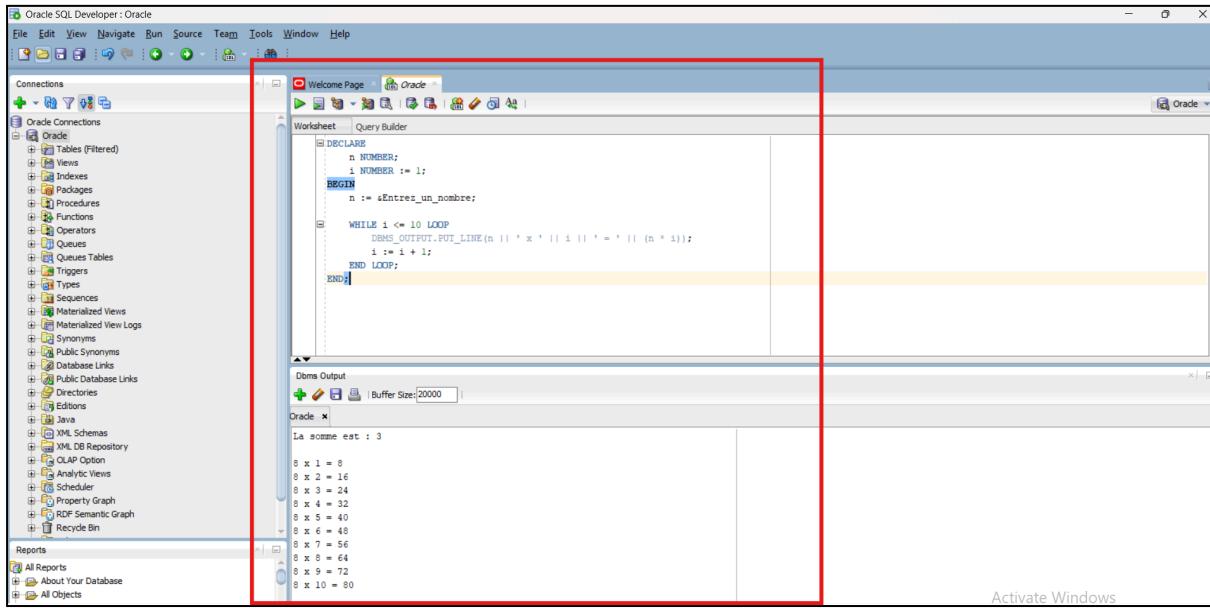
Below the worksheet is the "Doms Output" window, which contains the result of the execution:

```

La somme est : 3

```

2. Écrire une procédure anonyme PL/SQL qui permet d'afficher la table de multiplication d'un nombre.



```

DECLARE
    n NUMBER;
    i NUMBER := 1;
BEGIN
    n := &Entres_un_nombre;
    WHILE i <= 10 LOOP
        DBMS_OUTPUT.PUT_LINE(n || ' x ' || i || ' = ' || (n * i));
        i := i + 1;
    END LOOP;
END;

```

Doms Output
Buffer Size: 20000

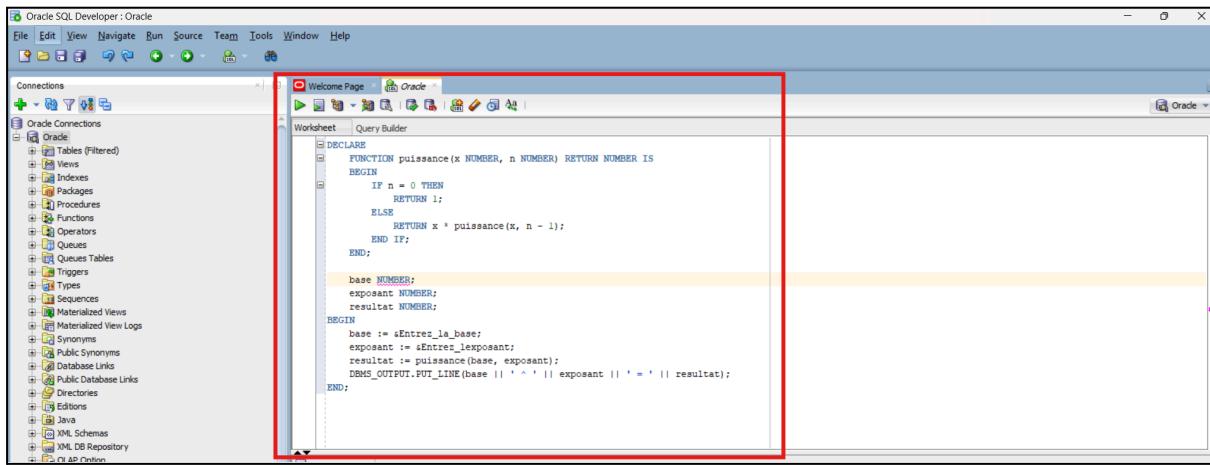
La somme est : 3

```

8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80

```

3. Écrire une fonction récursive PL/SQL qui retourne x^n avec deux entiers positifs.



```

FUNCTION puissance(x NUMBER, n NUMBER) RETURN NUMBER IS
BEGIN
    IF n = 0 THEN
        RETURN 1;
    ELSE
        RETURN x * puissance(x, n - 1);
    END IF;
END;

```

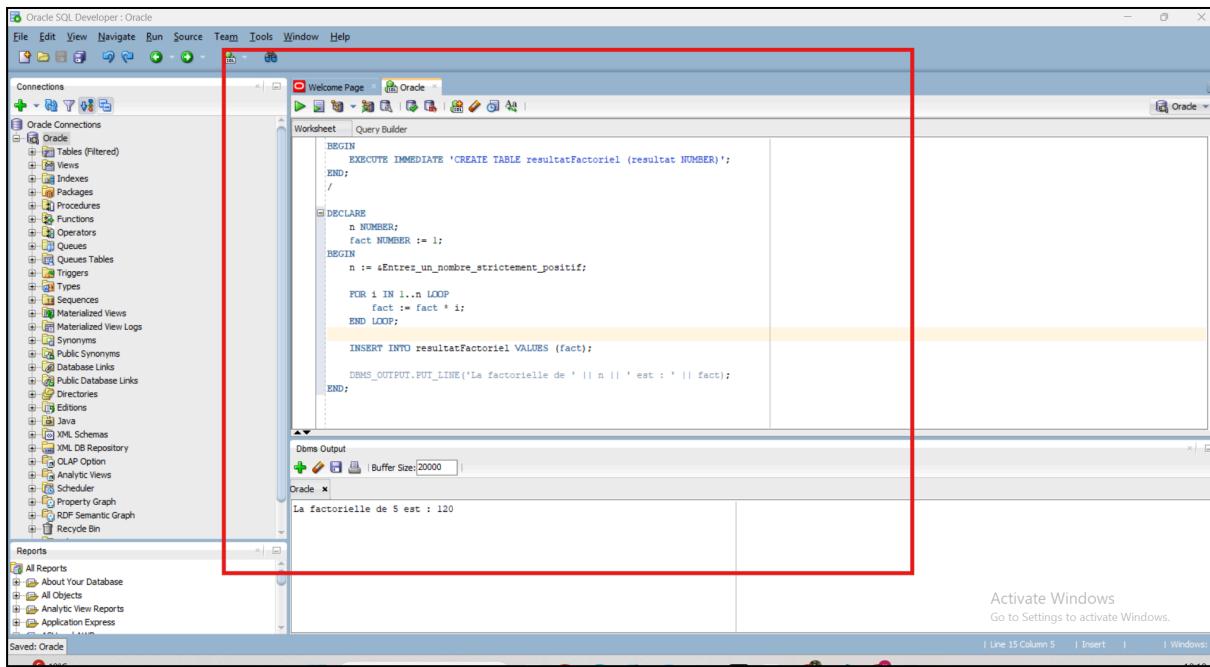
Doms Output

```

base NUMBER;
exposant NUMBER;
resultat NUMBER;
BEGIN
    base := &Entres_la_base;
    exposant := &Entret_exposant;
    resultat := puissance(base, exposant);
    DBMS_OUTPUT.PUT_LINE(base || ' ^ ' || exposant || ' = ' || resultat);
END;

```

4. Écrire une procédure anonyme PL/SQL pour calculer la factorielle d'un nombre saisi et stocker le résultat dans une table resultatFactoriel.



```

BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE resultatFactoriel (resultat NUMBER)';
END;
/

DECLARE
    n NUMBER;
    fact NUMBER := 1;
BEGIN
    n := <Entrez_un_nombre_strictement_positif>;
    FOR i IN 1..n LOOP
        fact := fact * i;
    END LOOP;

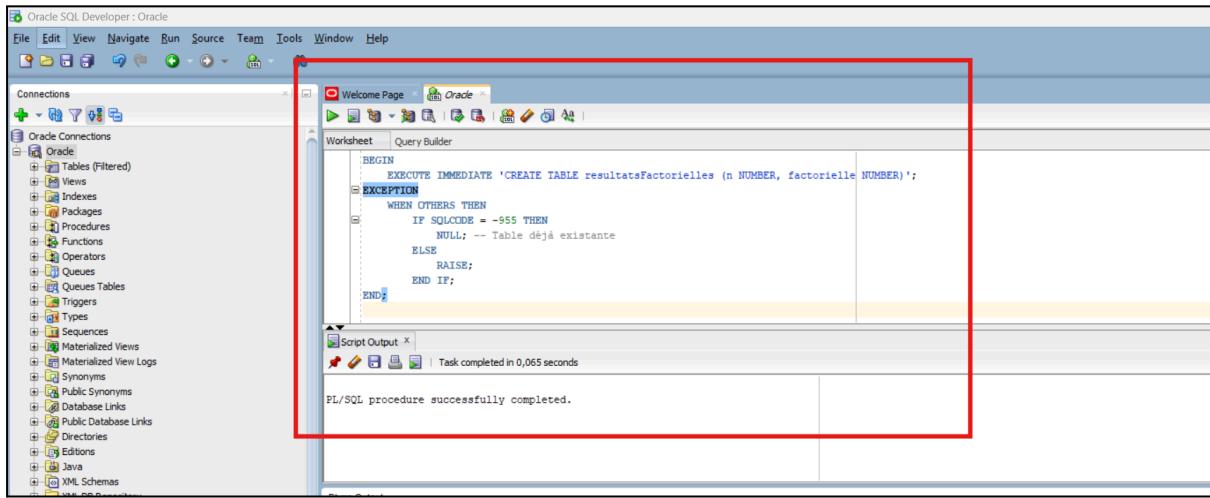
    INSERT INTO resultatFactoriel VALUES (fact);

    DBMS_OUTPUT.PUT_LINE('La factorielle de ' || n || ' est : ' || fact);
END;

```

The screenshot shows the Oracle SQL Developer interface. A red box highlights the code area in the Worksheet tab. The code creates a table 'resultatFactoriel' and then calculates the factorial of a user-specified number, printing the result to the DBMS_OUTPUT window.

5.Modifier le programme précédent pour stocker les factorielles des 20 premiers entiers dans resultatsFactorielles.



```

BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE resultatsFactorielles (n NUMBER, factorielle NUMBER)';
EXCEPTION
    WHEN OTHERS THEN
        IF SQLCODE = -955 THEN
            NULL; -- Table déjà existante
        ELSE
            RAISE;
        END IF;
END;

```

The screenshot shows the Oracle SQL Developer interface. A red box highlights the code area in the Worksheet tab. The code creates a table 'resultatsFactorielles' and handles an exception for a table already existing by doing nothing or raising an error.

The screenshot shows the Oracle SQL Developer interface. On the left, the Connections pane displays a single connection named 'Oracle'. The main area is a 'Worksheet' tab where the following PL/SQL code is written:

```
DECLARE
    n NUMBER := 1;
    fact NUMBER;
BEGIN
    WHILE n <= 20 LOOP
        fact := 1;
        FOR i IN 1..n LOOP
            fact := fact * i;
        END LOOP;
        INSERT INTO resultatsFactorielles (n, factorielle) VALUES (n, fact);
        DBMS_OUTPUT.PUT_LINE('Factorielle de ' || n || ' = ' || fact);
        n := n + 1;
    END LOOP;
END;
```

The screenshot shows the 'Dbms Output' window. It has a toolbar with icons for New, Edit, Save, Print, and Buffer Size set to 20000. The output area contains the results of the factorial calculation:

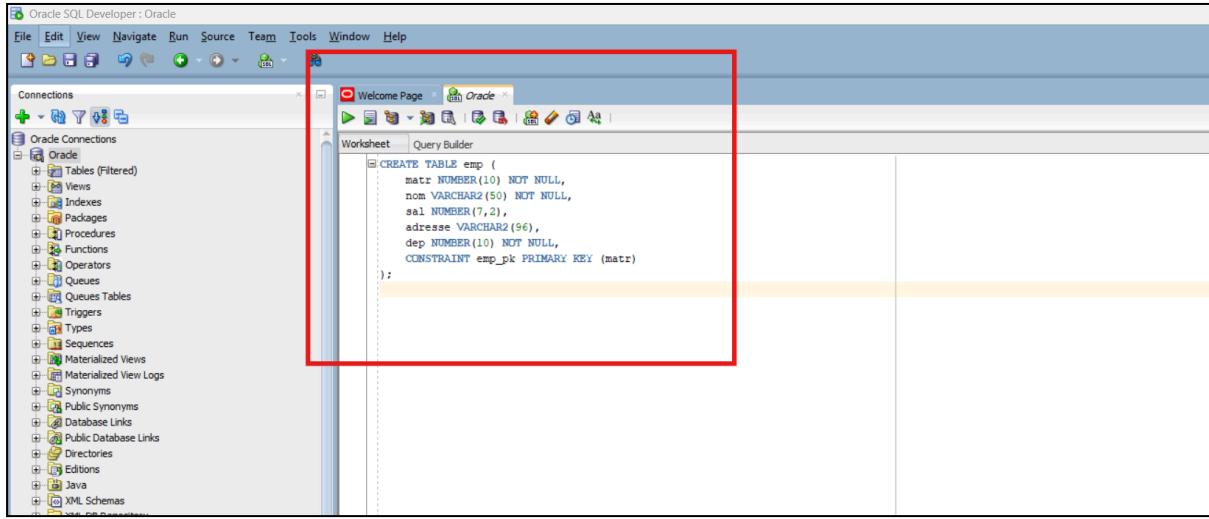
```
Factorielle de 1 = 1
Factorielle de 2 = 2
Factorielle de 3 = 6
Factorielle de 4 = 24
Factorielle de 5 = 120
Factorielle de 6 = 720
Factorielle de 7 = 5040
Factorielle de 8 = 40320
Factorielle de 9 = 362880
Factorielle de 10 = 3628800
Factorielle de 11 = 39916800
Factorielle de 12 = 479001600
Factorielle de 13 = 6227020800
Factorielle de 14 = 87178291200
Factorielle de 15 = 1307674368000
Factorielle de 16 = 20922789888000
Factorielle de 17 = 355687428096000
Factorielle de 18 = 6402373705728000
Factorielle de 19 = 121645100408832000
Factorielle de 20 = 2432902008176640000
```

Exercice 2 : Gestion d'une base de données employé

Créer une table emp représentant des employés, puis insérer, modifier et interroger ses données via des blocs PL/SQL.

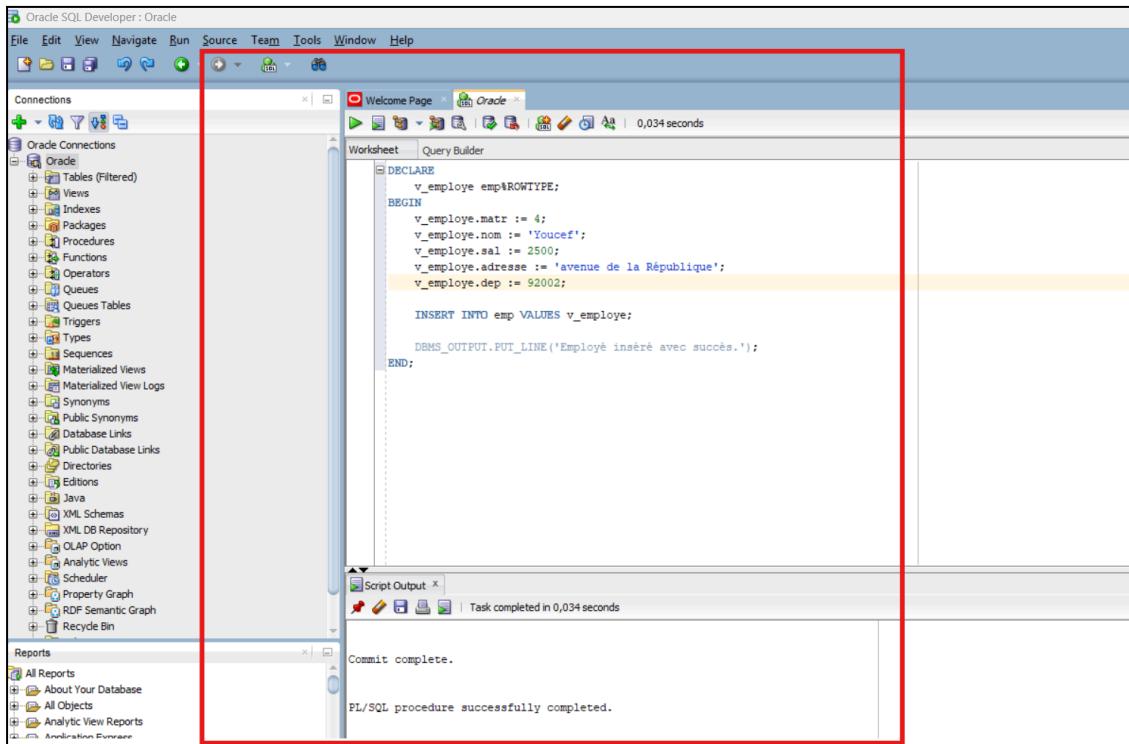
Création de la table emp

Afin de travailler sur la gestion des employés, nous avons commencé par créer la table emp en SQL :



```
CREATE TABLE emp (
    matr NUMBER(10) NOT NULL,
    nom VARCHAR(50) NOT NULL,
    sal NUMBER(7,2),
    adresse VARCHAR(96),
    dep NUMBER(10) NOT NULL,
    CONSTRAINT emp_pk PRIMARY KEY (matr)
);
```

Écrire un bloc anonyme en PL/SQL pour insérer un nouvel employé dans la table emp



```
DECLARE
    v_employe emp%ROWTYPE;
BEGIN
    v_employe.matr := 4;
    v_employe.nom := 'Youssef';
    v_employe.sal := 2500;
    v_employe.adresse := 'avenue de la République';
    v_employe.dep := 92002;

    INSERT INTO emp VALUES v_employe;

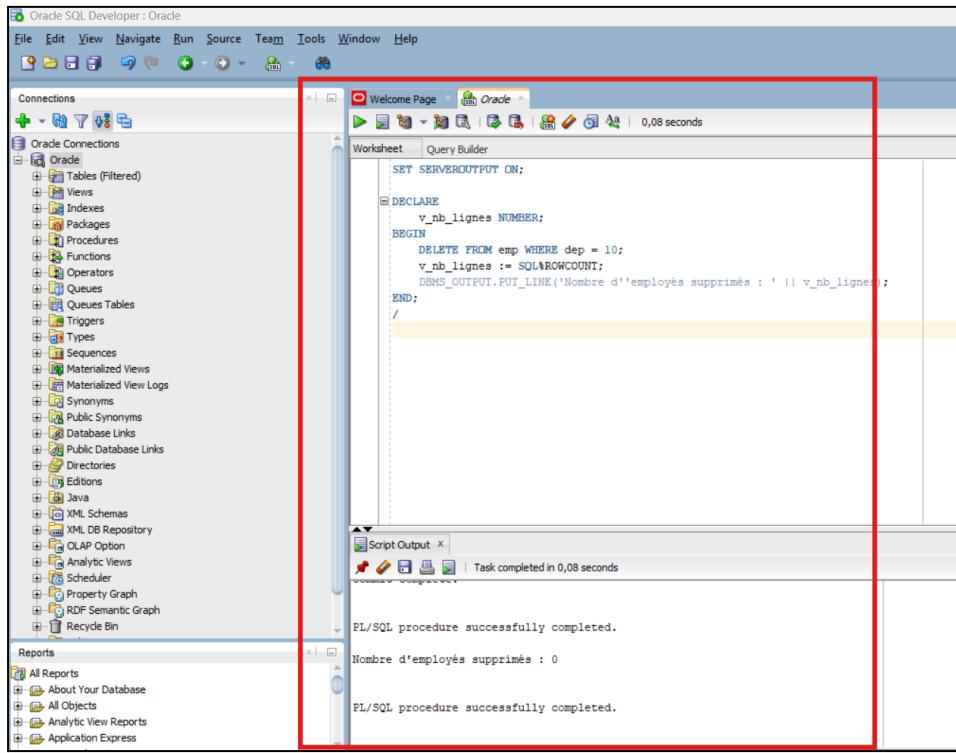
    DBMS_OUTPUT.PUT_LINE('Employé inséré avec succès.');
END;
```

Script Output X | Task completed in 0,034 seconds

Commit complete.

PL/SQL procedure successfully completed.

Suppression d'employés et affichage du nombre supprimé



```

SET SERVEROUTPUT ON;
DECLARE
    v_nb_lignes NUMBER;
BEGIN
    DELETE FROM emp WHERE dep = 10;
    v_nb_lignes := SQL%ROWCOUNT;
    DBMS_OUTPUT.PUT_LINE('Nombre d''employés supprimés : ' || v_nb_lignes);
END;
/

```

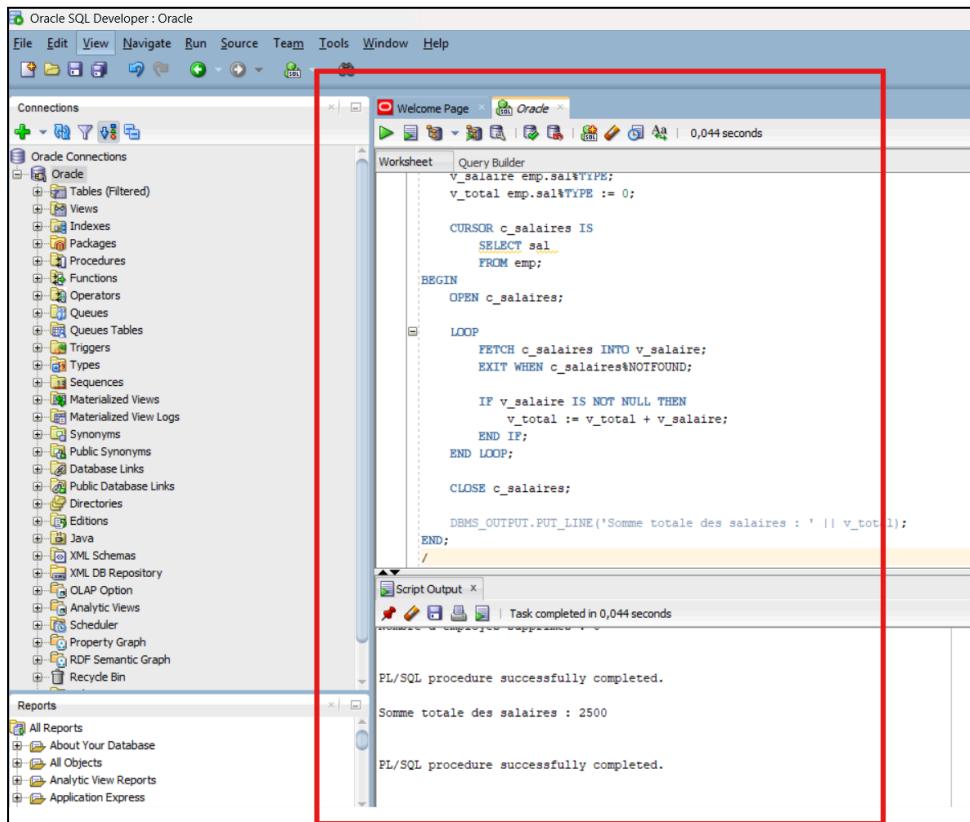
Script Output x | Task completed in 0,08 seconds

PL/SQL procedure successfully completed.

Nombre d'employés supprimés : 0

PL/SQL procedure successfully completed.

Afficher la somme des salaires avec un curseur explicite (boucle LOOP)



```

V_salaire emp.sal%TYPE;
v_total emp.sal%TYPE := 0;

CURSOR c_salaires IS
    SELECT sal
    FROM emp;
BEGIN
    OPEN c_salaires;
    LOOP
        FETCH c_salaires INTO v_salaire;
        EXIT WHEN c_salaires%NOTFOUND;
        IF v_salaire IS NOT NULL THEN
            v_total := v_total + v_salaire;
        END IF;
    END LOOP;
    CLOSE c_salaires;
    DBMS_OUTPUT.PUT_LINE('Somme totale des salaires : ' || v_total);
END;
/

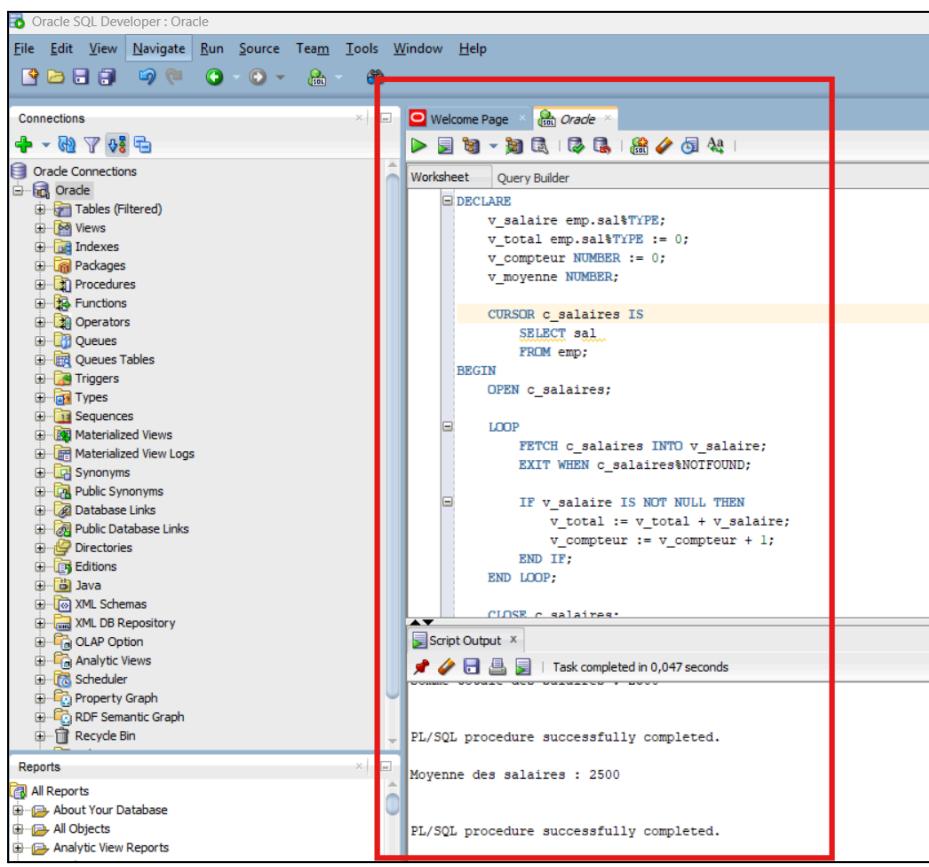
```

Script Output x | Task completed in 0,044 seconds

Somme totale des salaires : 2500

PL/SQL procedure successfully completed.

Calculer la moyenne des salaires (en modifiant la procédure précédente)



```

DECLARE
    v_salaire emp.sal%TYPE;
    v_total emp.sal%TYPE := 0;
    v_compteur NUMBER := 0;
    v_moyenne NUMBER;

    CURSOR c_salaires IS
        SELECT sal...
        FROM emp;

    BEGIN
        OPEN c_salaires;

        LOOP
            FETCH c_salaires INTO v_salaire;
            EXIT WHEN c_salaires%NOTFOUND;

            IF v_salaire IS NOT NULL THEN
                v_total := v_total + v_salaire;
                v_compteur := v_compteur + 1;
            END IF;
        END LOOP;

        CLOSE c_salaires;
    END;
  
```

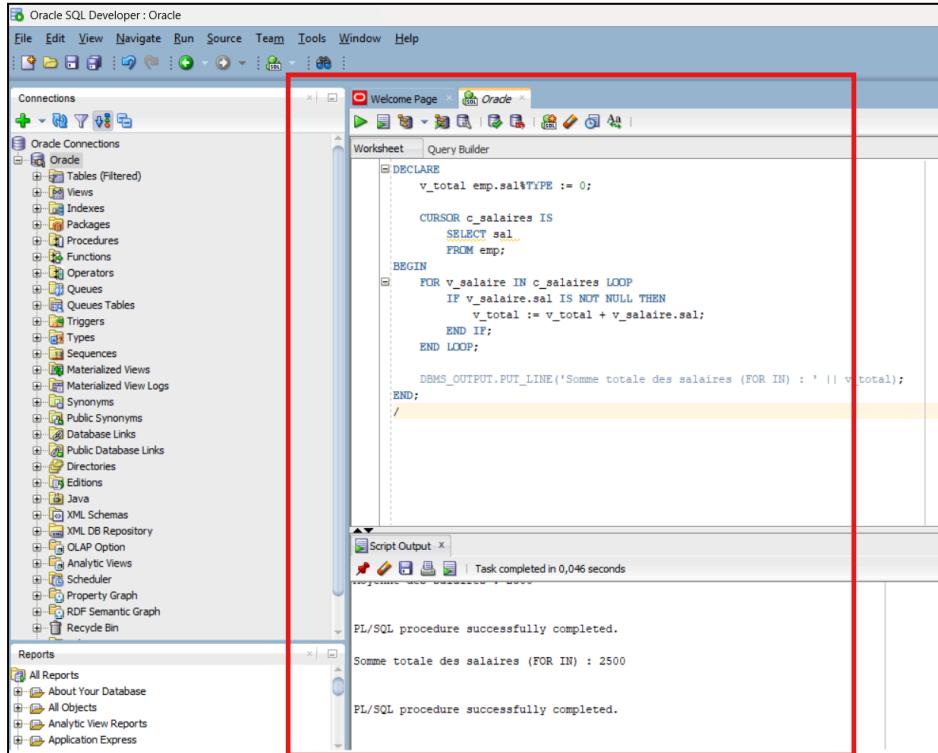
Script Output x | Task completed in 0,047 seconds

PL/SQL procedure successfully completed.

Moyenne des salaires : 2500

PL/SQL procedure successfully completed.

Utiliser une boucle FOR IN au lieu du LOOP classique



```

DECLARE
    v_total emp.sal%TYPE := 0;

    CURSOR c_salaires IS
        SELECT sal...
        FROM emp;

    BEGIN
        FOR v_salaire IN c_salaires LOOP
            IF v_salaire.sal IS NOT NULL THEN
                v_total := v_total + v_salaire.sal;
            END IF;
        END LOOP;

        DBMS_OUTPUT.PUT_LINE('Somme totale des salaires (FOR IN) : ' || v_total);
    END;
  
```

Script Output x | Task completed in 0,046 seconds

PL/SQL procedure successfully completed.

Somme totale des salaires (FOR IN) : 2500

PL/SQL procedure successfully completed.

Afficher les noms d'employés d'un département donné avec un curseur paramétré

The screenshot shows the Oracle SQL Developer interface. On the left, the Connections sidebar shows an Oracle connection named 'Oracle'. The main workspace contains a 'Worksheet' tab with the following PL/SQL code:

```
DECLARE
    CURSOR c(p_dep emp.dep%TYPE) IS
        SELECT nom...
        FROM emp
        WHERE dep = p_dep;
BEGIN
    FOR v_employe IN c(92000) LOOP
        DBMS_OUTPUT.PUT_LINE('Employé du département 92000 : ' || v_employe.nom);
    END LOOP;

    FOR v_employe IN c(75000) LOOP
        DBMS_OUTPUT.PUT_LINE('Employé du département 75000 : ' || v_employe.nom);
    END LOOP;
END;
/
```

Below the worksheet, the 'Script Output' tab displays three messages indicating successful completion of the procedure:

- PL/SQL procedure successfully completed.
- PL/SQL procedure successfully completed.
- PL/SQL procedure successfully completed.

Exercice 3 :

Avant tout, il faut une table client (si elle n'existe pas encore) :

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a 'Welcome Page' tab and an 'Oracle' tab. Below the tabs is a toolbar with various icons. The main area is titled 'Worksheet' and contains a query builder window. Inside the query builder, the following SQL code is written:

```
CREATE TABLE client (
    id_client NUMBER PRIMARY KEY,
    nom_client VARCHAR2(50),
    adresse_client VARCHAR2(100)
);
```

Below the worksheet is a 'Script Output' window. It displays the results of the execution:

```
PL/SQL procedure successfully completed.  
PL/SQL procedure successfully completed.  
Table CLIENT created.
```

Création du package specification

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a 'Connections' sidebar with a tree view of database objects. The main workspace is titled 'Worksheet' and contains the following PL/SQL code:

```

CREATE OR REPLACE PACKAGE gestion_clients IS
    -- Première procédure : ajoute un client avec id et nom
    PROCEDURE ajouter_client(p_id NUMBER, p_nom VARCHAR2);

    -- Deuxième procédure surchargée : ajoute un client avec id, nom et adresse
    PROCEDURE ajouter_client(p_id NUMBER, p_nom VARCHAR2, p_adresse VARCHAR2);
END gestion_clients;
/

```

In the bottom-right panel, the 'Script Output' window displays the results of the execution:

```

PL/SQL procedure successfully completed.

Table CLIENT created.

Package GESTION_CLIENTS compiled

```

Création du package body

This screenshot shows the Oracle SQL Developer interface with a red box highlighting the 'Worksheet' area where the package body is being defined. The 'Connections' sidebar on the left shows various database objects like Tables, Views, and Procedures.

The 'Worksheet' pane contains the following PL/SQL code:

```

CREATE OR REPLACE PACKAGE BODY gestion_clients IS
    -- Première version de la procédure
    PROCEDURE ajouter_client(p_id NUMBER, p_nom VARCHAR2) IS
    BEGIN
        INSERT INTO client(id_client, nom_client) VALUES (p_id, p_nom);
        DBMS_OUTPUT.PUT_LINE('Client inséré sans adresse.');
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Erreur : Client déjà existant.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Erreur inattendue : ' || SQLERRM);
    END;

    -- Deuxième version de la procédure (surcharge)
    PROCEDURE ajouter_client(p_id NUMBER, p_nom VARCHAR2, p_adresse VARCHAR2) IS
    BEGIN
        INSERT INTO client(id_client, nom_client, adresse_client) VALUES (p_id, p_nom, p_adresse);
        DBMS_OUTPUT.PUT_LINE('Client inséré avec adresse.');
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Erreur : Client déjà existant.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Erreur inattendue : ' || SQLERRM);
    END;
END gestion_clients;
/

```

The 'Script Output' window at the bottom right shows the successful compilation of the package body:

```

Task completed in 0,077 seconds

Package GESTION_CLIENTS compiled

```

Tester l'insertion simple :

The screenshot shows the Oracle SQL Developer interface. On the left, the Connections tree shows an Oracle connection named 'Oracle'. The central workspace contains a 'Worksheet' tab with the following PL/SQL code:

```
BEGIN
    gestion_clients.ajouter_client(1, 'Alice');
END;
/
```

Below the worksheet is a 'Script Output' window displaying the results of the execution:

```
Client inséré sans adresse.

PL/SQL procedure successfully completed.
```

A red box highlights the entire central workspace area.

Tester l'insertion avec adresse :

The screenshot shows the Oracle SQL Developer interface. On the left, the Connections tree shows an Oracle connection named 'Oracle'. The central workspace contains a 'Worksheet' tab with the following PL/SQL code:

```
BEGIN
    gestion_clients.ajouter_client(2, 'Bob', '10 rue de Paris');
END;
/
```

Below the worksheet is a 'Script Output' window displaying the results of the execution:

```
Client inséré sans adresse.

PL/SQL procedure successfully completed.

Client inséré avec adresse.

PL/SQL procedure successfully completed.
```

A red box highlights the entire central workspace area.

Conclusion

Ce TP nous a permis de pratiquer le PL/SQL à travers la création de procédures, l'utilisation de curseurs, la gestion d'exceptions et la réalisation d'un package, renforçant ainsi notre maîtrise de la programmation en base de données Oracle.