

COMPTE-RENDU TP4 BDA

EXERCICE 1 :

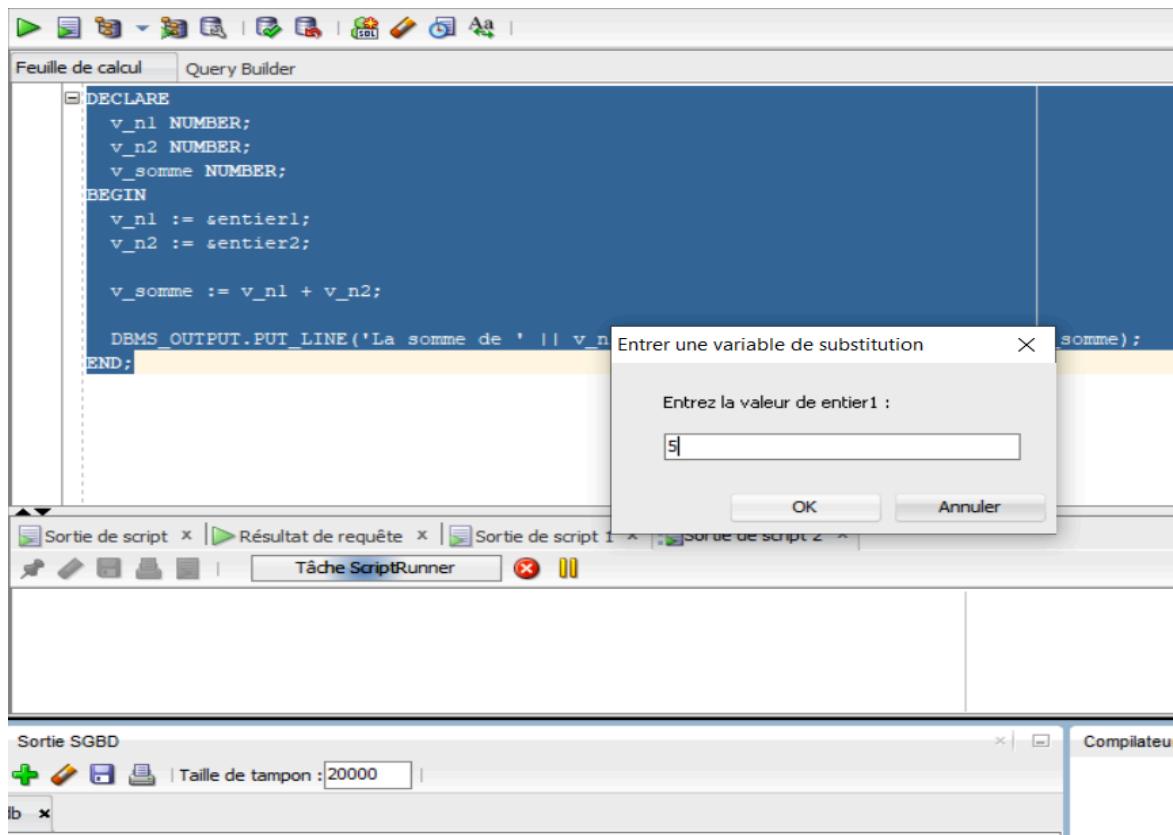
1/

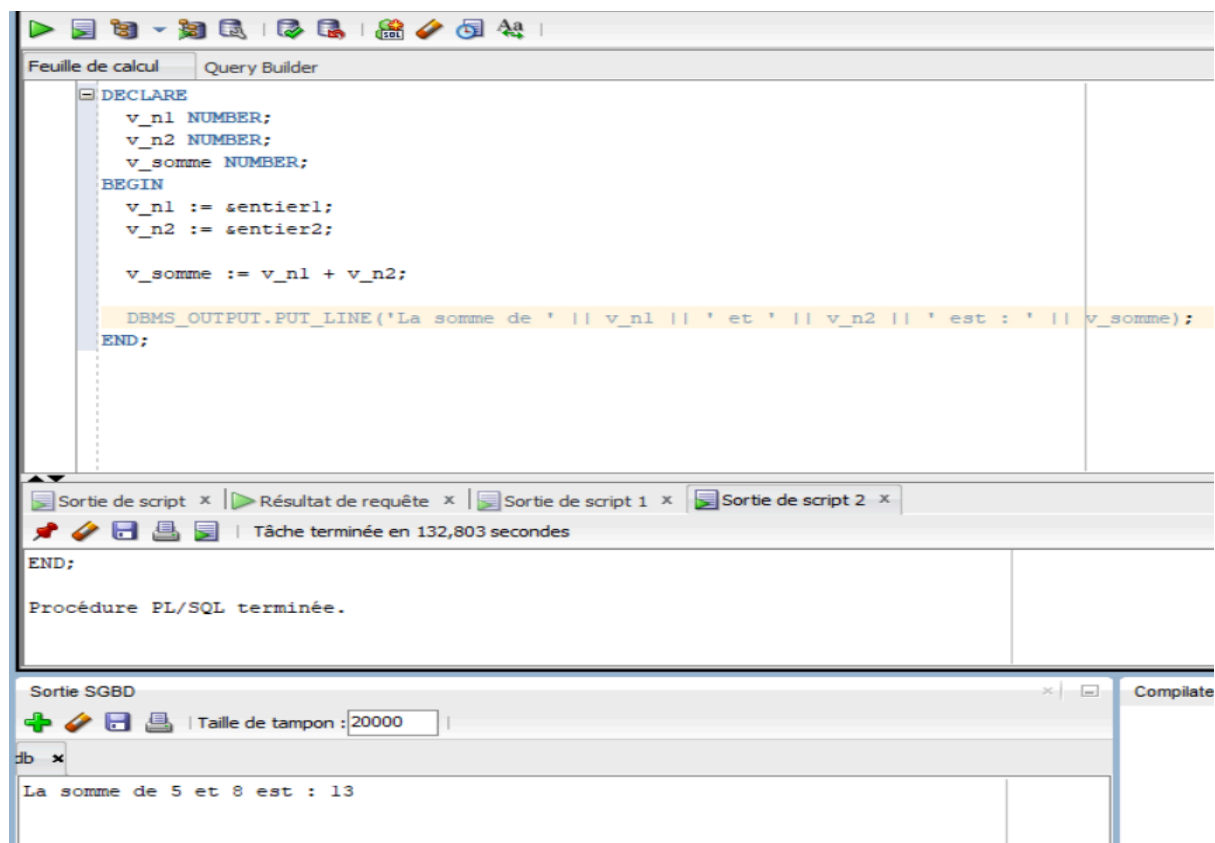
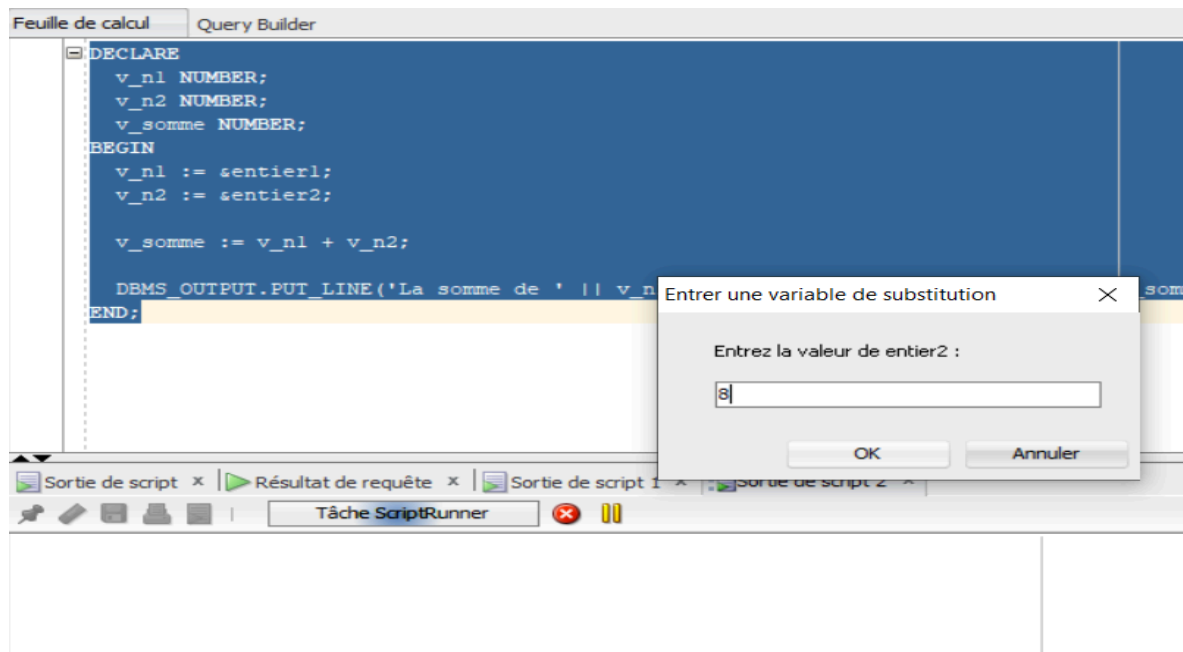
On déclare 3 entiers v_n1, v_n2 et v_somme. On lit les deux entiers à l'aide de &entier1 et &entier2.

Ensuite, on réalise la somme qui sera stockée dans la variable v_somme. Ensuite, on affiche la somme de v_n1 et de v_n2 au niveau de la console à l'aide de DBMS_OUTPUT.PUT_LINE.

```
DECLARE
    v_v_n1 NUMBER;
    v_n2 NUMBER;
    v_somme NUMBER;
BEGIN
    v_n1 := &entier1;
    v_n2 := &entier2;
    v_somme := v_n1 + v_n2;
    DBMS_OUTPUT.PUT_LINE('La v_somme de ' || v_n1 || ' et ' || v_n2 || ' est : ' || v_somme);
END;
```

Suite à l'exécution deux fenêtres seront affichées l'une après l'autre pour renseigner les valeurs de v_n1 et v_n2.





2/

Tout d'abord, on crée deux variables `v_nombre` et `v_resultat`. `v_nombre` va contenir un nombre qui sera renseigné par l'utilisateur. Suite à cela, on affiche un message indiquant qu'il s'agit de la table de multiplication du nombre `v_nombre`. Ensuite, on fait une boucle de 1 à 10 au niveau de laquelle on affiche la multiplication du nombre `v_nombre * i` et `v_resultat` qui représente la variable au niveau de laquelle on va stocker le résultat. On gère notamment les exceptions à la fin du code pour s'assurer du fait qu'on puisse bien gérer les éventuelles erreurs.

```

DECLARE
    v_nombre NUMBER;
    v_resultat NUMBER;
BEGIN
    v_nombre := &nbr;

    DBMS_OUTPUT.PUT_LINE('Table de multiplication de ' || v_nombre || '
:');

    FOR i IN 1..10 LOOP
        v_resultat := v_nombre * i;
        DBMS_OUTPUT.PUT_LINE(v_nombre || ' x ' || i || ' = ' ||
v_resultat);
    END LOOP;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur : ' || SQLERRM);
END;

```

On obtient le résultat suivant :

The screenshot shows the SQL Developer interface. The top pane displays the PL/SQL script. The bottom pane shows the execution results in the 'Sortie SGBD' window. The task was completed in 6.152 seconds.

Sortie SGBD

Table de multiplication de 9 :

```

9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90

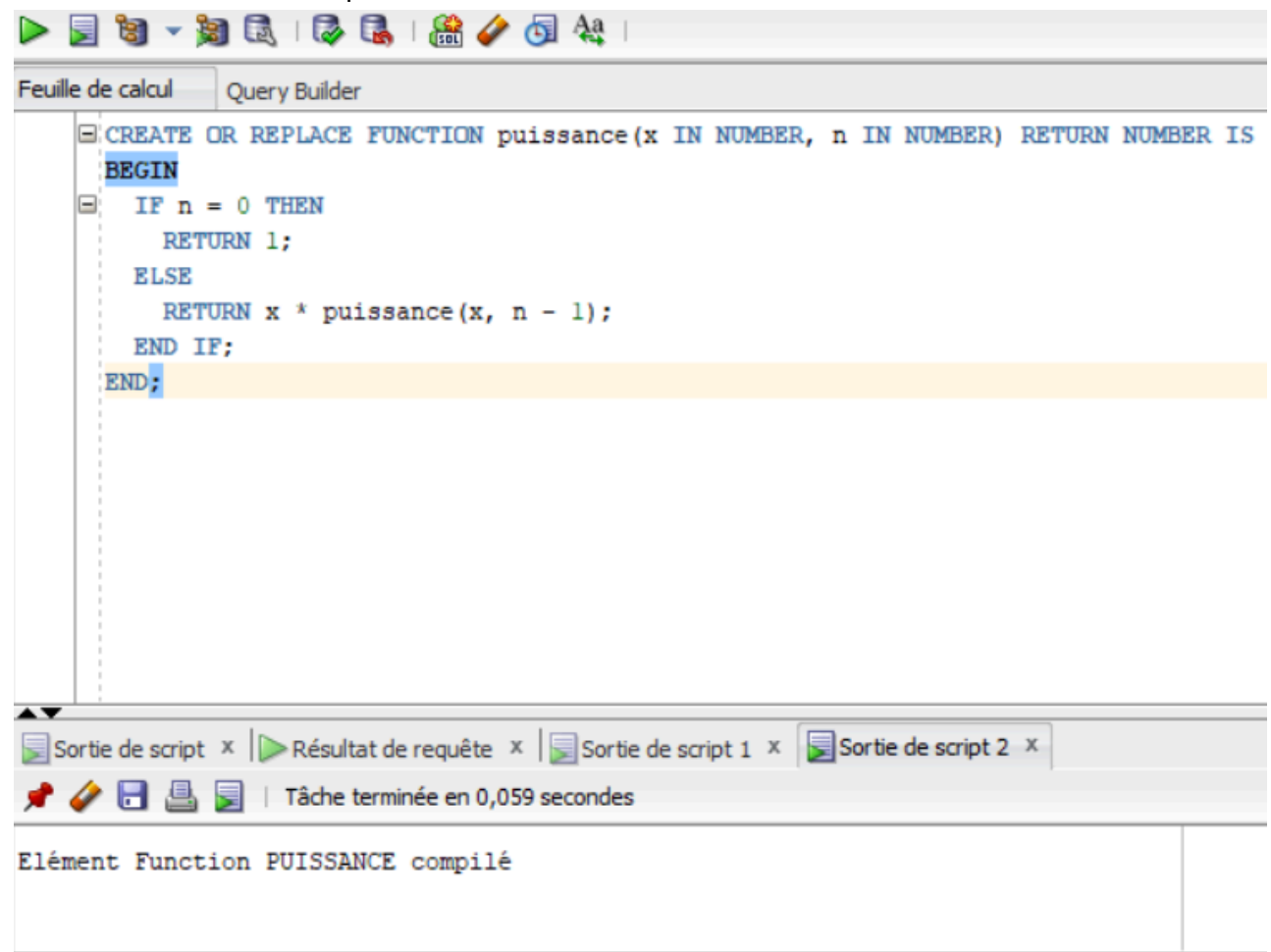
```

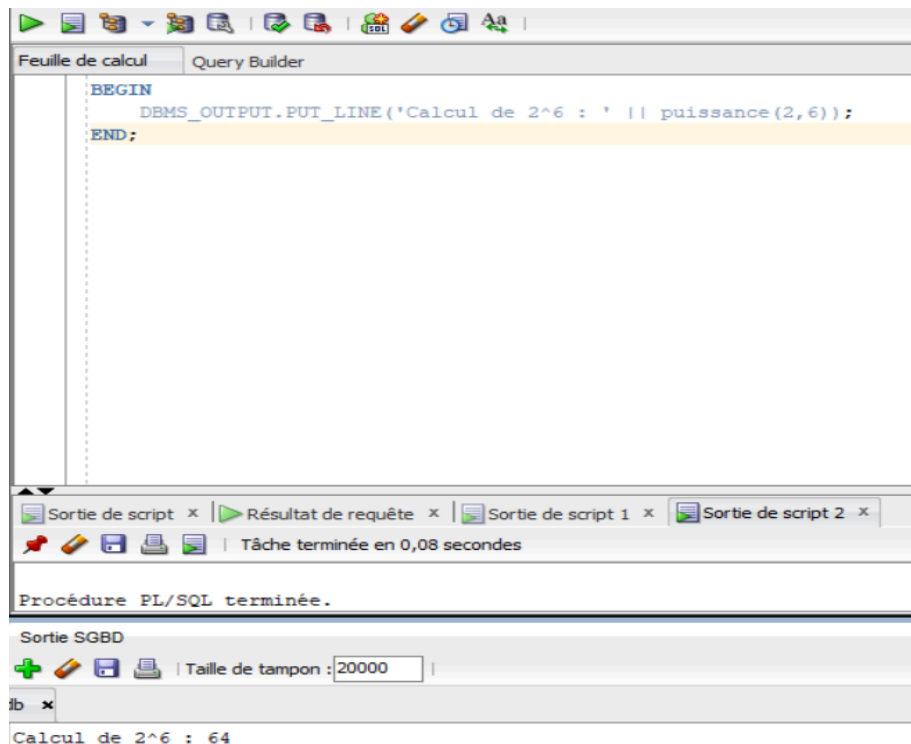
3/

Tout d'abord, on va créer une fonction tout en écrasant la fonction existante avec REPLACE si cette fonction existe. La fonction prend deux paramètres: un entier x et un entier n. On notera l'utilisation de IN qui représente d'une input de la fonction. La fonction retourne un entier. Le cas de base est $n = 0$, dans ce cas on retourne 1 (x^0). Sinon, on va retourner la multiplication de x par la puissance(x,n-1). Il s'agit ici de l'appel récursif. On décrémente l'exposant et on multiplie par la valeur de l'entier. Ainsi, on trouve le résultat recherché dès qu'on atteint le cas de base.

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

On va déclarer la fonction par la suite. On va tester son exécution.





4/Tout d'abord, on va créer une table factorielle qui contiendra le nombre et la factorielle associée.

```
CREATE TABLE resultatFactoriel (
  nombre    NUMBER PRIMARY KEY,
  factoriel  NUMBER
);
```

Ensuite, on écrit une procédure qui va stocker le résultat dans cette table.

La procédure va déclarer deux variables `v_nombre` et `v_fact`. `v_fact` sera initialisée à 1.

`v_nombre` correspondra à une entrée utilisateur. Si l'entrée saisie par l'utilisateur est négative, on demandera à l'utilisateur de saisir un nombre positif (la factorielle ne peut être calculée que sur des nombres positifs). Ensuite, on calcule la factorielle de manière itérative en effectuant une boucle de 1 à l'entrée utilisateur et en enchaînant des multiplications qui seront stockées dans la variable `v_fact`. On insère par la suite le résultat dans `resultatFactoriel`. Suite à cela, on affiche le résultat du calcul. On lèvera notamment des exceptions dans le cas où on tente de stocker une même factorielle de fois et on affiche un message d'erreur pour toute erreur inconnue.

```
DECLARE
  v_nombre NUMBER;
  v_fact   NUMBER := 1;
BEGIN
  v_nombre := &nb;

  IF v_nombre <= 0 THEN
    DBMS_OUTPUT.PUT_LINE('Erreur : veuillez saisir un nombre
strictement positif.');
```

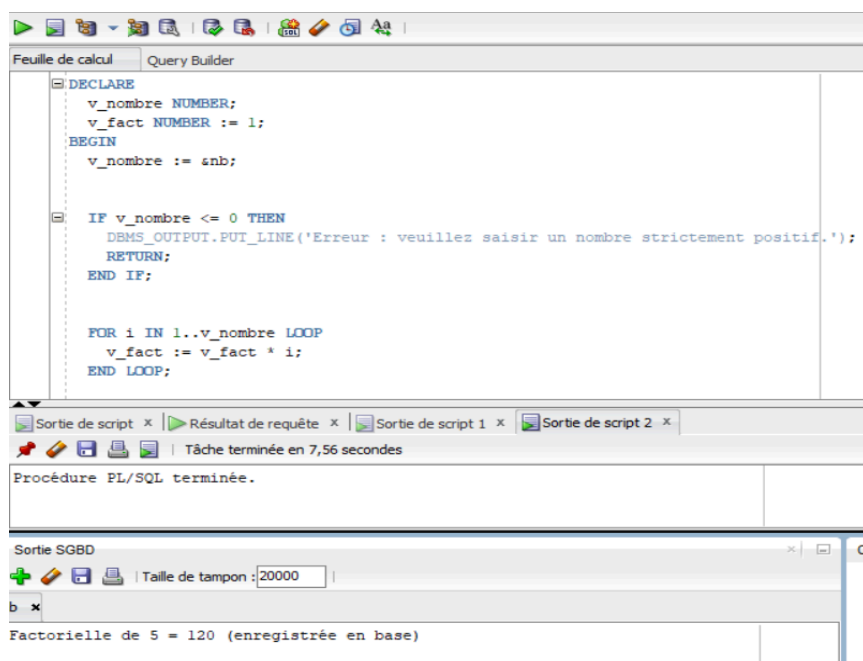
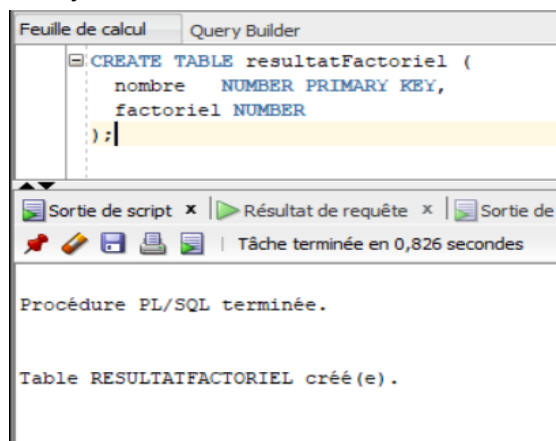
```

FOR i IN 1..v_nombre LOOP
    v_fact := v_fact * i;
END LOOP;
INSERT INTO resultatFactoriel(nombre, factoriel)
VALUES (v_nombre, v_fact);
DBMS_OUTPUT.PUT_LINE('Factorielle de ' || v_nombre || ' = ' || v_fact
|| ' (enregistrée en base)');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('La factorielle de ce nombre a déjà été
enregistrée.');
```

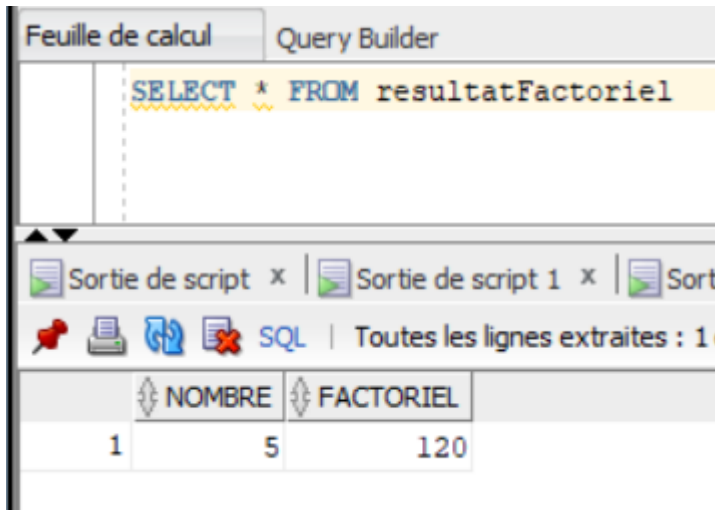
```

    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur : ' || SQLERRM);
END;
```

On crée tout d'abord la table dans la base de données. Après cela on crée la procédure anonyme.



On vérifie que la valeur a bien été enregistrée dans la base de données.



On crée une nouvelle table resultatsFactoriels.

```
CREATE TABLE resultatsFactoriels (  
  nombre      NUMBER PRIMARY KEY,  
  factoriel   NUMBER  
);
```

On utilisera le même code précédent mais on ajoutera une boucle pour réaliser l'opération sur les 20 premiers entiers. On réinitialise v_fact à chaque itération et on itère de 1 à i pour trouver chaque factorielle. On devra notamment créer une table ResultatsFactoriels pour stocker les résultats.

```
DECLARE  
  v_fact NUMBER := 1;  
BEGIN  
  FOR i IN 1..20 LOOP  
    v_fact := 1;  
    FOR j IN 1..i LOOP  
      v_fact := v_fact * j;  
    END LOOP;  
    INSERT INTO resultatsFactoriels(nombre, factoriel)  
    VALUES (i, v_fact);  
  
    DBMS_OUTPUT.PUT_LINE(i || '!' = ' || v_fact);  
  END LOOP;  
EXCEPTION  
  WHEN DUP_VAL_ON_INDEX THEN  
    DBMS_OUTPUT.PUT_LINE('Certains résultats sont déjà présents dans la  
table.');
```

```
  WHEN OTHERS THEN  
    DBMS_OUTPUT.PUT_LINE('Erreur : ' || SQLERRM);  
END;
```

On crée tout d'abord la table.

```
CREATE TABLE resultatsFactoriels (  
    nombre    NUMBER PRIMARY KEY,  
    factoriel  NUMBER  
);
```

Sortie de script x | Sortie de script 1 x

Tâche terminée en 0,472 secondes

Table RESULTATSFACRIELS créé(e).

Par la suite, on exécute la procédure anonyme ci-dessus.

```
FOR i IN 1..20 LOOP  
    v_fact := 1;  
    FOR j IN 1..i LOOP  
        v_fact := v_fact * j;  
    END LOOP;  
    INSERT INTO resultatsFactoriels(nombre, factoriel)
```

Sortie de script x | Sortie de script 1 x

Tâche terminée en 0,137 secondes

Table RESULTATSFACRIELS créé(e).

Procédure PL/SQL terminée.

Sortie SGBD

Taille de tampon : 20000

db x

```
3! = 6  
4! = 24  
5! = 120  
6! = 720  
7! = 5040  
8! = 40320  
9! = 362880  
10! = 3628800  
11! = 39916800  
12! = 479001600  
13! = 6227020800  
14! = 87178291200  
15! = 1307674368000  
16! = 20922789888000  
17! = 355687428096000  
18! = 6402373705728000
```

On vérifie par la suite l'ajout des entiers à la table.

Feuille de calcul			
Query Builder			
SELECT * FROM resultatsFactoriels;			
Sortie de script x Sortie de script 1 x Résultat de requête x			
SQL Toutes les lignes extraites : 20 en 0,015 secondes			
	NOMBRE	FACTORIEL	
1	1	1	
2	2	2	
3	3	6	
4	4	24	
5	5	120	
6	6	720	
7	7	5040	
8	8	40320	
9	9	362880	
10	10	3628800	
11	11	39916800	
12	12	479001600	
13	13	6227020800	
14	14	87178291200	
15	15	1307674368000	
16	16	20922789888000	
17	17	355687428096000	
18	18	6402373705728000	
19	19	121645100408832000	
20	20	2432902008176640000	

EXERCICE 2 :

1/On crée tout d'abord la table Employé.

```
CREATE TABLE emp (
  matr NUMBER(10) NOT NULL,
  nom VARCHAR2(50) NOT NULL,
  sal NUMBER(7,2),
  adresse VARCHAR2(96),
  dep NUMBER(10) NOT NULL,
  CONSTRAINT emp_pk PRIMARY KEY (matr)
);
```

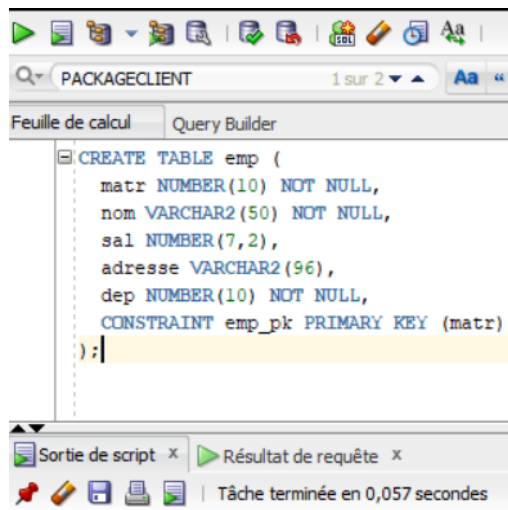


Table EMP créé(e).

Par la suite, on crée le bloc anonyme PL/SQL permettant d'ajouter l'utilisateur souhaité. On notera que les exceptions ont été gérées dans le cas où un utilisateur avec le même ID soit inséré ou même au cas où une autre erreur se produit.

```
DECLARE
    v_employe emp%ROWTYPE;
BEGIN
    v_employe.matr := 4;
    v_employe.nom := 'Youcef';
    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('Insertion réussie de l''employé.');
```

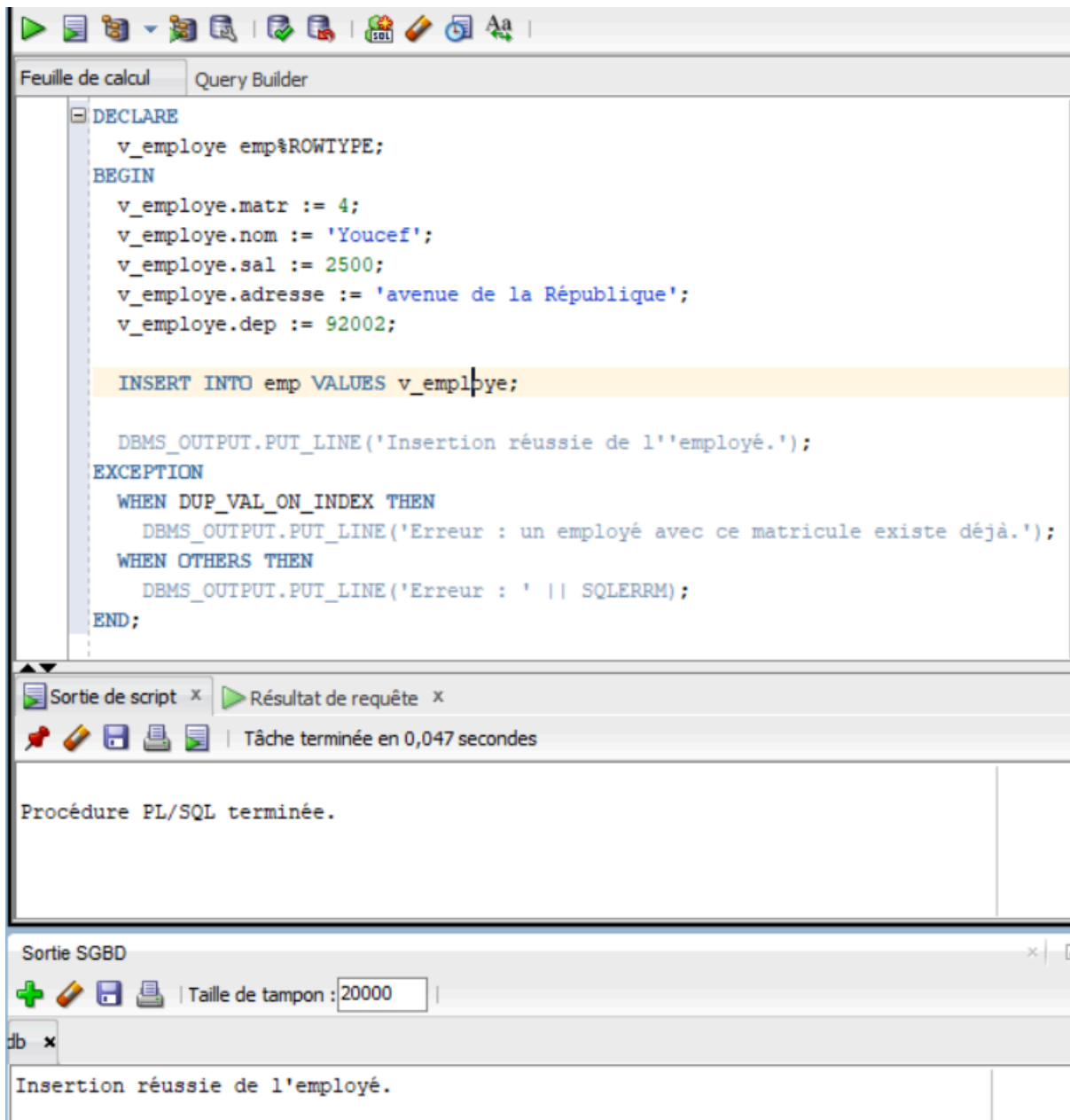
EXCEPTION

```
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Erreur : un employé avec ce matricule existe déjà.');
```

WHEN OTHERS THEN

```
        DBMS_OUTPUT.PUT_LINE('Erreur : ' || SQLERRM);
END;
```

On exécute par la suite ce code:



2/

```
BEGIN
    DELETE FROM emp
    WHERE dep IS NOT NULL;

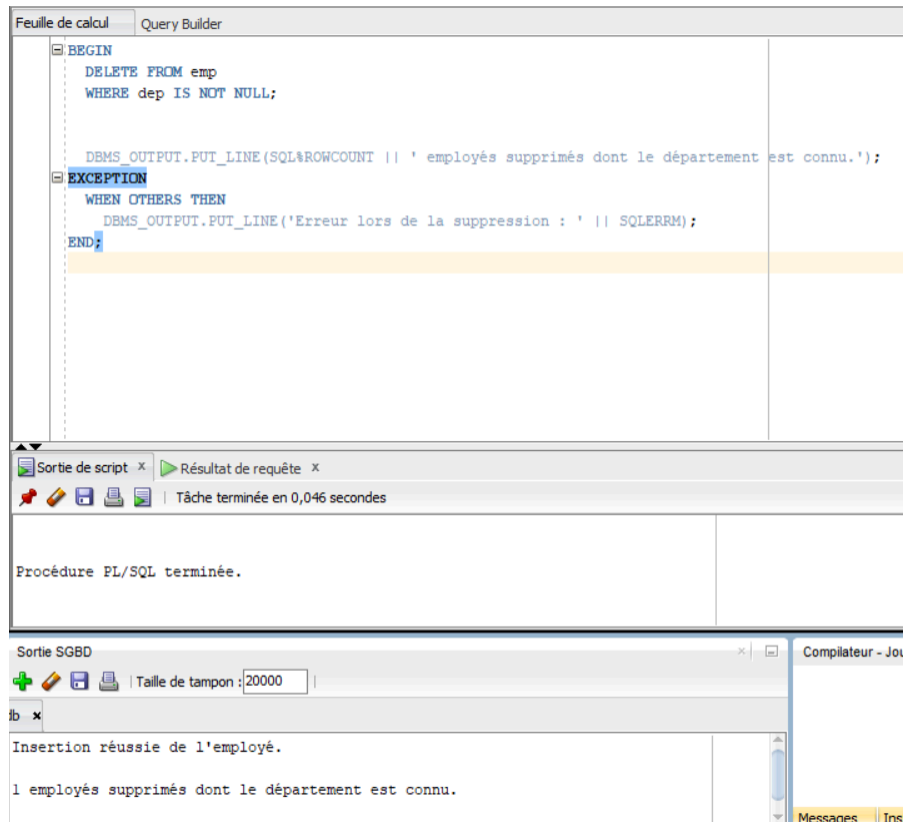
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' employés supprimés dont le
département est connu.');
```

Exception block (partially visible):

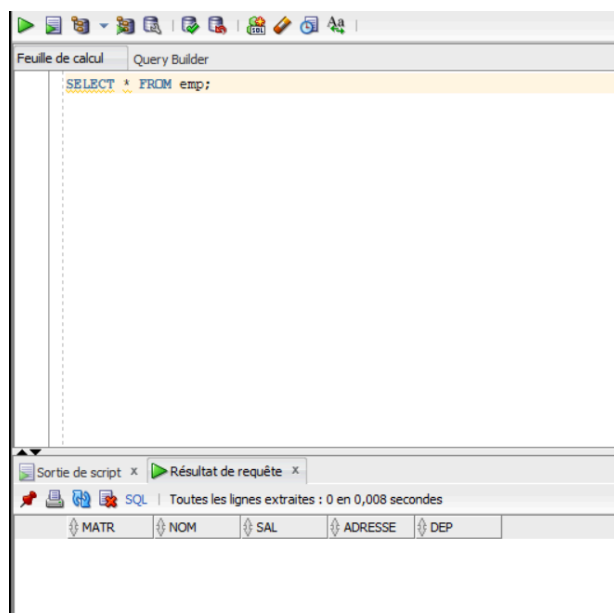
```
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur lors de la suppression : ' ||
SQLERRM);
```

END;

Pour cette question, on réalise la suppression des utilisateurs dont le département est connu. Suite à cela, on utilise la variable SQL%ROWCOUNT pour obtenir le nombre de lignes supprimées dans la table Employé. Dans notre cas, on a seulement l'employé inséré dans la question 1. Donc, puisqu'il n'a pas de département affecté. Il sera supprimé.



On vérifie la suppression en faisant un select * sur la table employé. On trouvera ainsi que la table est vide.

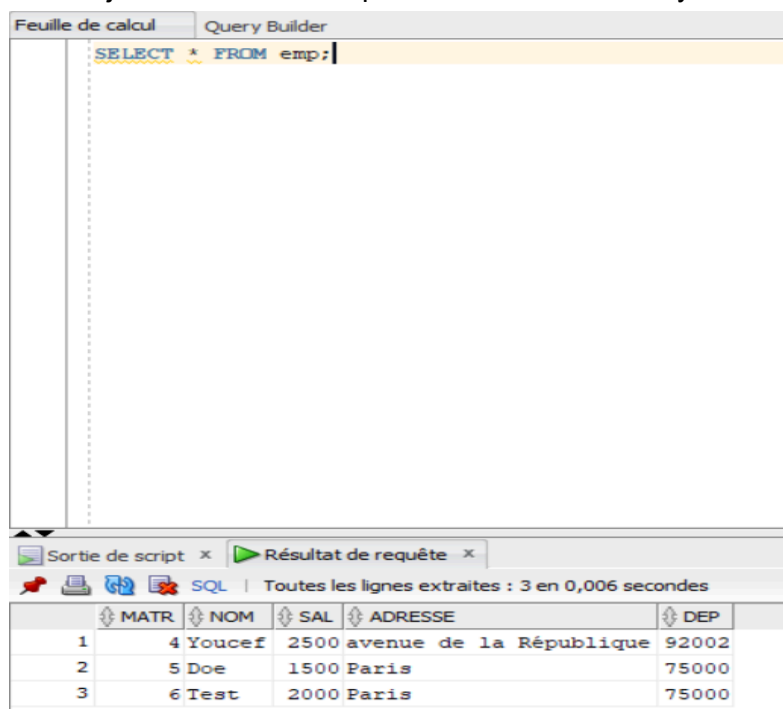


3/ Pour calculer la somme des salaires, on crée un curseur contenant l'ensemble des salaires des employés. Après cela, on déclare une variable emp.sal ayant le type du champ salaire de la table emp. Par la suite, on crée une variable v_somme qui va contenir la

somme de l'ensemble des salaires. Ensuite, on ouvre un curseur nommé c_emp et on parcourt ce curseur à l'aide d'une boucle LOOP, on utilise FETCH pour mettre le contenu du salaire au niveau du curseur dans la variable v_sal. On s'arrête quand le curseur ne contient plus de salaire. A chaque itération, on ajoute le salaire à la variable v_somme. Après la fin de la boucle. On affiche le contenu de la variable v_somme. On traite notamment le cas des exceptions.

```
DECLARE
    CURSOR c_emp IS
        SELECT sal FROM emp;
    v_sal emp.sal%TYPE;
    v_somme NUMBER := 0;
BEGIN
    OPEN c_emp;
    LOOP
        FETCH c_emp INTO v_sal;
        EXIT WHEN c_emp%NOTFOUND;
        v_somme := v_somme + v_sal;
    END LOOP;
    CLOSE c_emp;
    DBMS_OUTPUT.PUT_LINE('Somme des salaires des employés : ' ||
v_somme);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur lors du calcul : ' || SQLERRM);
END;
```

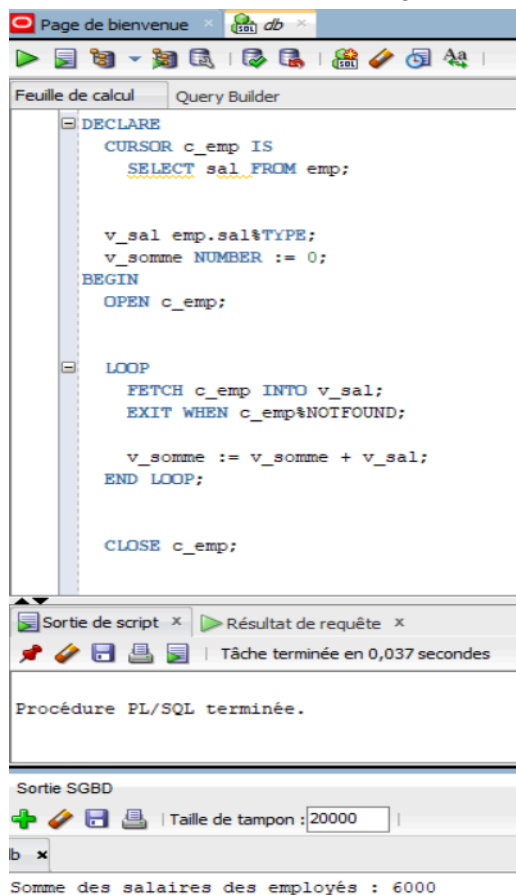
On va ajouter 3 utilisateurs pour tester ce bloc anonyme.



The screenshot shows a database interface with a 'Query Builder' window and a 'Résultat de requête' (Query Result) window. The query builder contains the SQL statement 'SELECT * FROM emp;'. The query result window displays a table with 3 rows of data, including columns for MATR (ID), NOM (Name), SAL (Salary), ADRESSE (Address), and DEP (Department).

	MATR	NOM	SAL	ADRESSE	DEP
1	4	Youcef	2500	avenue de la République	92002
2	5	Doe	1500	Paris	75000
3	6	Test	2000	Paris	75000

On utilise par la suite le bloc anonyme pour vérifier la somme obtenue:
On trouve que la somme est égale à 6000. Ce qui est correct.



4/ Pour calculer le salaire moyen des employés, on doit ajouter une nouvelle variable `v_nb` qui nous permettra de compter le nombre total d'employés. Par la suite, on divise la somme par le nombre d'employés.

```
DECLARE
  CURSOR c_emp IS
    SELECT sal FROM emp;

  v_sal emp.sal%TYPE;
  v_somme NUMBER := 0;
  v_nb NUMBER := 0;
BEGIN
  OPEN c_emp;

  LOOP
    FETCH c_emp INTO v_sal;
    EXIT WHEN c_emp%NOTFOUND;

    v_somme := v_somme + v_sal;

```

```

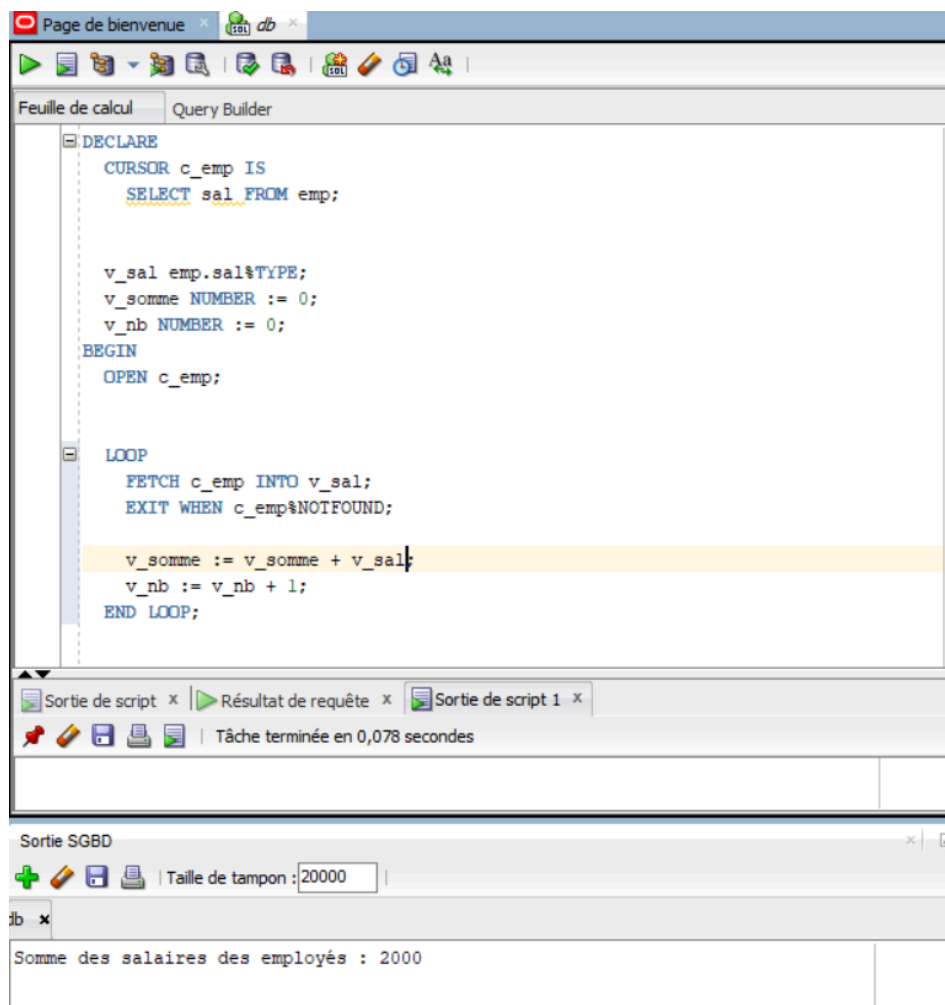
        v_nb := v_nb + 1;
    END LOOP;

    CLOSE c_emp;
    IF v_nb > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Somme des salaires des employés : ' ||
v_somme/v_nb);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Aucun salaire renseigné.');
```

```

    END IF;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur lors du calcul : ' || SQLERRM);
END;
```

En réalisant, l'exécution pour les mêmes employés présents dans la base de données, on trouve le résultat suivant:



Pour obtenir le salaire total de tous les employés. On devra juste faire un select de l'ensemble des salaires à partir de la base de données. Par la suite, on parcourt le résultat de la requête en utilisant la boucle FOR...IN grâce à la variable emp_sal.

Les autres traitements resteront similaires au code précédent.

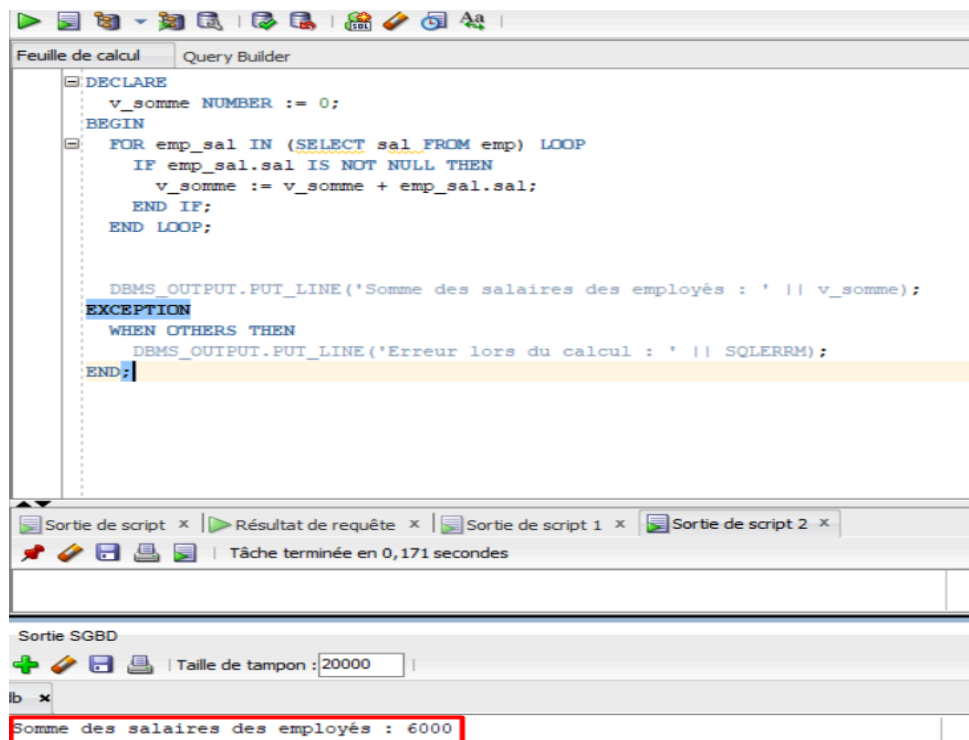
On notera que pour obtenir le salaire à partir de la variable emp_sal. On utilise emp_rec.sal.

On devra par la suite ajouter le salaire à v_somme au niveau de la boucle. Par la suite, on affiche v_somme.

```
DECLARE
    v_somme NUMBER := 0;
BEGIN
    FOR emp_sal IN (SELECT sal FROM emp) LOOP
        IF emp_sal.sal IS NOT NULL THEN
            v_somme := v_somme + emp_sal.sal;
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Somme des salaires des employés : ' ||
v_somme);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur lors du calcul : ' || SQLERRM);
END;
```

On obtient ainsi le résultat suivant:



Pour la deuxième requête, c'est la même idée. On devra juste utiliser une autre variable v_nb pour compter des employés existants.


```

DECLARE
    v_somme NUMBER := 0;
    v_nb NUMBER := 0;
    v_moyenne NUMBER;
BEGIN
    FOR emp_sal IN (SELECT sal FROM emp) LOOP
        IF emp_sal.sal IS NOT NULL THEN
            v_somme := v_somme + emp_sal.sal;
            v_nb := v_nb + 1;
        END IF;
    END LOOP;

    IF v_nb > 0 THEN
        v_moyenne := v_somme / v_nb;
        DBMS_OUTPUT.PUT_LINE('Salaire moyen des employés : ' || v_moyenne);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Aucun salaire renseigné.');
```

Pour la moyenne, ce sera la même idée que le bloc anonyme précédent. Seulement, on devra ajouter une variable v_nb qui compte le nombre d'employés. Cette variable sera utilisée pour calculer la moyenne par la suite.

```

DECLARE
    v_somme NUMBER := 0;
    v_nb NUMBER := 0;
    v_moyenne NUMBER;
BEGIN
    FOR emp_sal IN (SELECT sal FROM emp) LOOP
        IF emp_sal.sal IS NOT NULL THEN
            v_somme := v_somme + emp_sal.sal;
            v_nb := v_nb + 1;
        END IF;
    END LOOP;

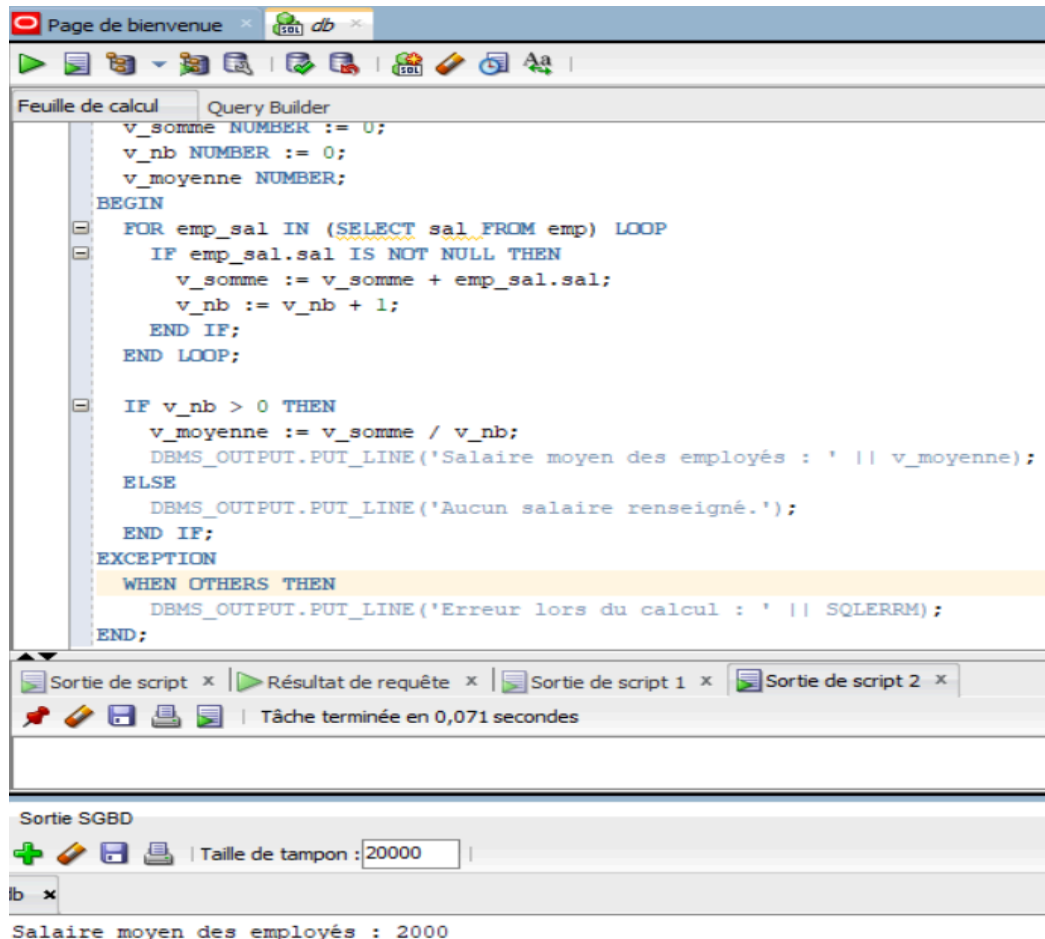
    IF v_nb > 0 THEN
        v_moyenne := v_somme / v_nb;
        DBMS_OUTPUT.PUT_LINE('Salaire moyen des employés : ' || v_moyenne);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Aucun salaire renseigné.');
```

```

END IF;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Erreur lors du calcul : ' || SQLERRM);
END;

```

Suite à l'exécution, on obtient le résultat suivant:



6/

Tout d'abord, on va déclarer un curseur `c_emp` qui contiendra le nom des employés présents dans le département passé en paramètre. Suite à cela, on va créer une variable `v_nom` qui aura le même type que le champ `nom` de la table `emp`. Ensuite, on va ouvrir un curseur pour le code postal 92000. On va réaliser une boucle pour parcourir ce curseur. On prendra ainsi le contenu du curseur et on le chargera dans la variable `v_nom` grâce à `FETCH`, par la suite on va afficher cette variable. Ce processus sera répété jusqu'à ce que le curseur soit vide. Les mêmes étapes seront réalisées pour le code postal 75000. On gèrera aussi les exceptions à la fin du code.

```

DECLARE
  CURSOR c_emp(p_dep NUMBER) IS
    SELECT nom FROM emp WHERE dep = p_dep;

  v_nom emp.nom%TYPE;

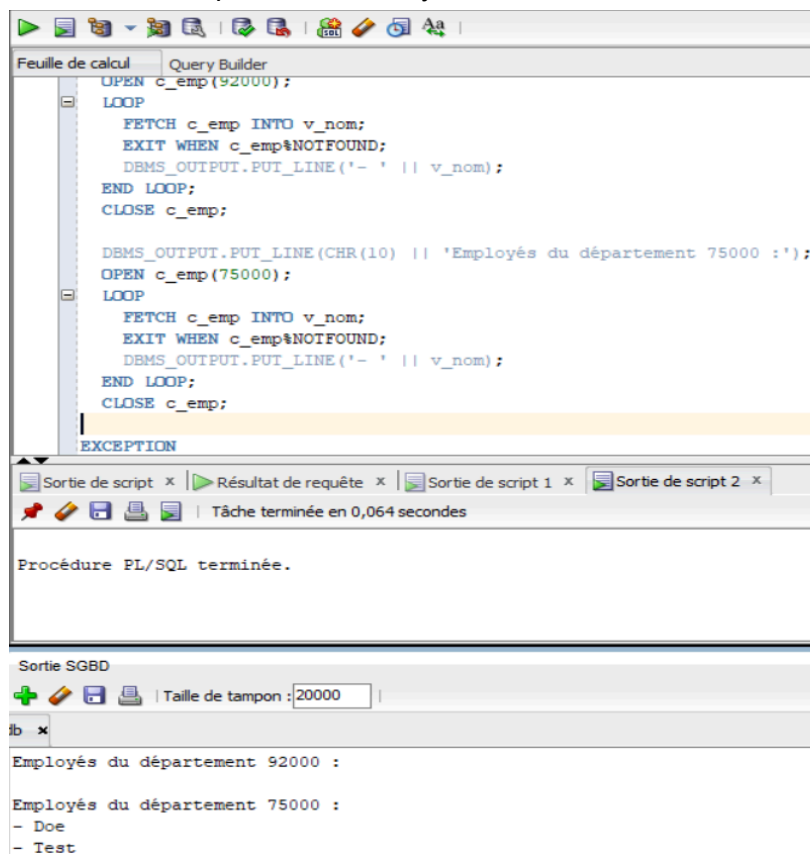
```

```

BEGIN
    DBMS_OUTPUT.PUT_LINE('Employés du département 92000 :');
    OPEN c_emp(92000);
    LOOP
        FETCH c_emp INTO v_nom;
        EXIT WHEN c_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('- ' || v_nom);
    END LOOP;
    CLOSE c_emp;
    DBMS_OUTPUT.PUT_LINE(chr(10) || 'Employés du département 75000 :');
    OPEN c_emp(75000);
    LOOP
        FETCH c_emp INTO v_nom;
        EXIT WHEN c_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('- ' || v_nom);
    END LOOP;
    CLOSE c_emp;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur : ' || SQLERRM);
END;

```

En exécutant la procédure anonyme, on obtient le résultat suivant:



EXERCICE 3:

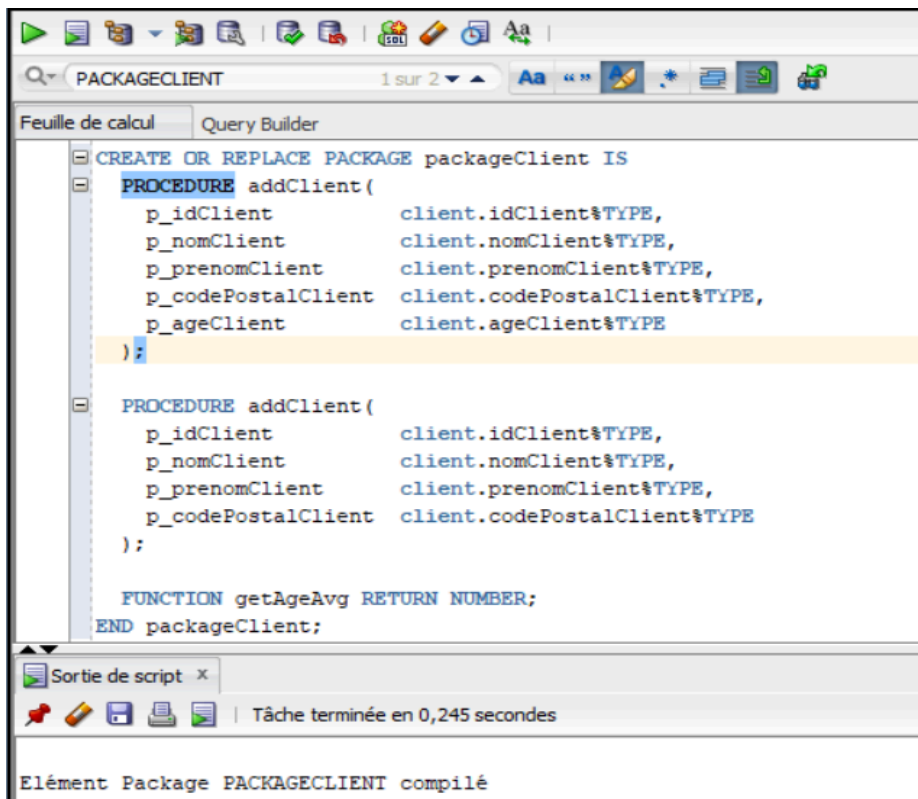
On crée tout d'abord la spécification:

```
CREATE OR REPLACE PACKAGE packageClient IS
    PROCEDURE addClient(
        p_idClient          client.idClient%TYPE,
        p_nomClient         client.nomClient%TYPE,
        p_prenomClient      client.prenomClient%TYPE,
        p_codePostalClient  client.codePostalClient%TYPE,
        p_ageClient         client.ageClient%TYPE
    );

    PROCEDURE addClient(
        p_idClient          client.idClient%TYPE,
        p_nomClient         client.nomClient%TYPE,
        p_prenomClient      client.prenomClient%TYPE,
        p_codePostalClient  client.codePostalClient%TYPE
    );

    FUNCTION getAgeAvg RETURN NUMBER;
END packageClient;
```

Ensuite, on compile le code à l'aide de SQLDeveloper. Le code est compilé avec succès.



Par la suite, on implémente cette spécification:

```
CREATE OR REPLACE PACKAGE BODY packageClient IS
```

```

PROCEDURE addClient(

    p_idClient      client.idClient%TYPE,

    p_nomClient      client.nomClient%TYPE,

    p_prenomClient   client.prenomClient%TYPE,

    p_codePostalClient client.codePostalClient%TYPE,

    p_ageClient      client.ageClient%TYPE

) IS

BEGIN

    INSERT INTO client(idClient, nomClient, prenomClient, codePostalClient, ageClient)

    VALUES (p_idClient, p_nomClient, p_prenomClient, p_codePostalClient, p_ageClient);

EXCEPTION

    WHEN DUP_VAL_ON_INDEX THEN

        DBMS_OUTPUT.PUT_LINE('Erreur lors de la création : Cet utilisateur existe déjà !');

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('Erreur Inattendue ! : ' || SQLERRM);

END;

```

```

PROCEDURE addClient(

    p_idClient      client.idClient%TYPE,

    p_nomClient      client.nomClient%TYPE,

    p_prenomClient   client.prenomClient%TYPE,

    p_codePostalClient client.codePostalClient%TYPE

) IS

```

```
BEGIN

INSERT INTO client(idClient, nomClient, prenomClient, codePostalClient, ageClient)

VALUES (p_idClient, p_nomClient, p_prenomClient, p_codePostalClient, NULL);

EXCEPTION

WHEN DUP_VAL_ON_INDEX THEN

    DBMS_OUTPUT.PUT_LINE('Erreur lors de la création : Cet utilisateur existe déjà !');

WHEN OTHERS THEN

    DBMS_OUTPUT.PUT_LINE('Erreur Inattendue : ' || SQLERRM);

END;


FUNCTION getAgeAvg RETURN NUMBER IS

v_ageAvg NUMBER;

BEGIN

SELECT AVG(ageClient) INTO v_ageAvg FROM client WHERE ageClient IS NOT NULL;

RETURN v_ageAvg;

EXCEPTION

WHEN NO_DATA_FOUND THEN

    RETURN NULL;

WHEN OTHERS THEN

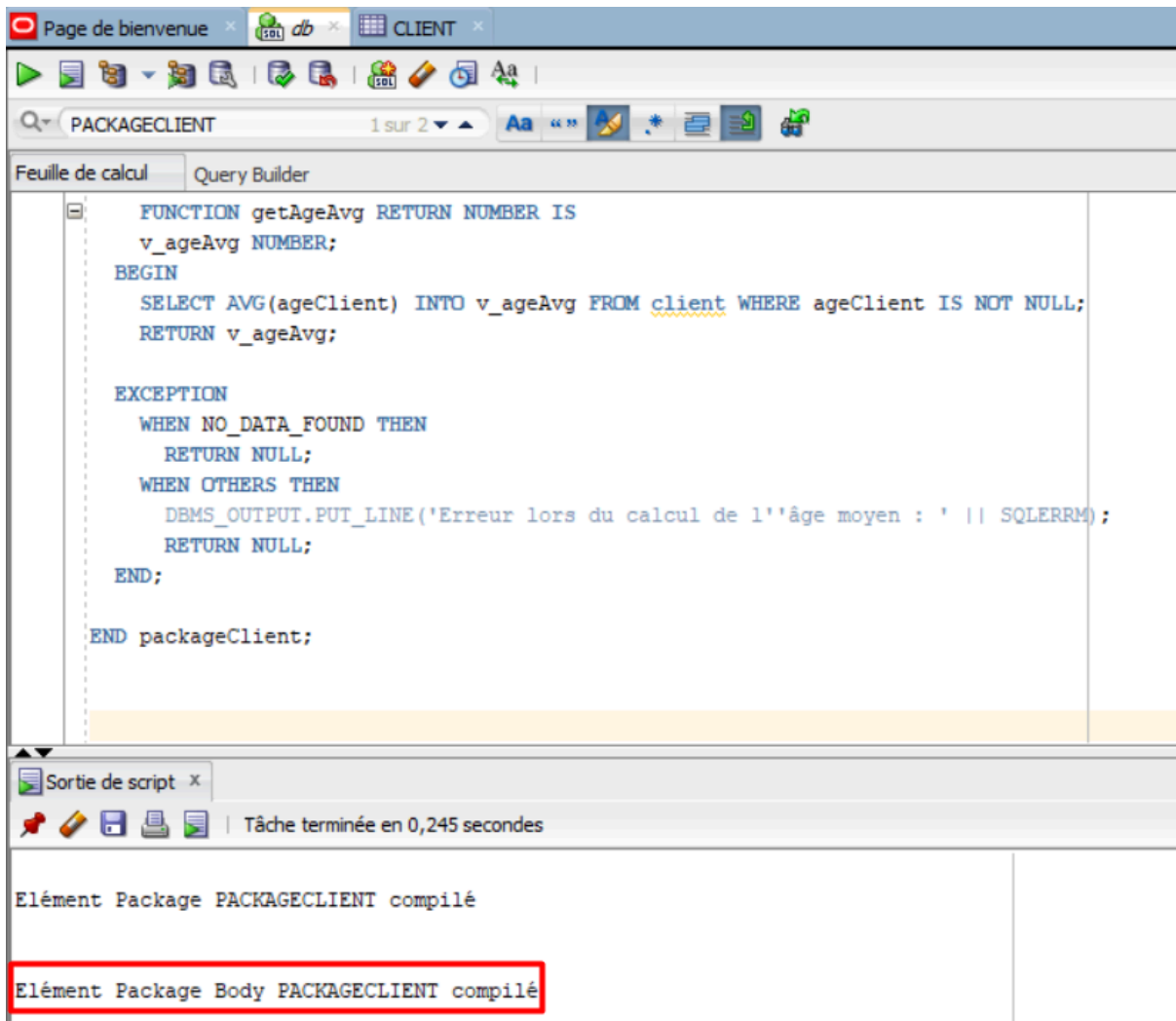
    DBMS_OUTPUT.PUT_LINE('Erreur lors du calcul de l'âge moyen : ' || SQLERRM);

    RETURN NULL;

END;
```

```
END packageClient;
```

On compile par la suite ce code.



Le code est compilé avec succès.

A ce niveau, on tente d'ajouter des clients au niveau de la base de données en utilisant les méthodes développées. Tout d'abord, on va créer un client qui disposera d'un âge.

```
BEGIN

packageClient.addClient(

  p_idClient => 1,

  p_nomClient => 'SADDEM',

  p_prenomClient => 'Elyes',
```

```

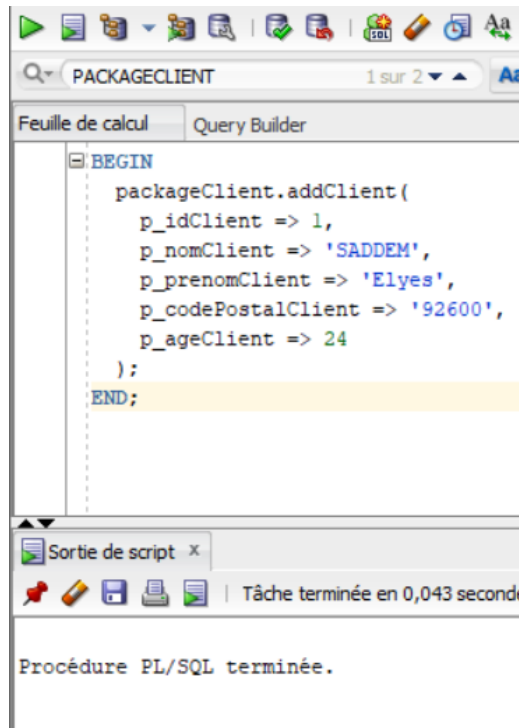
p_codePostalClient => '92600',

p_ageClient => 24

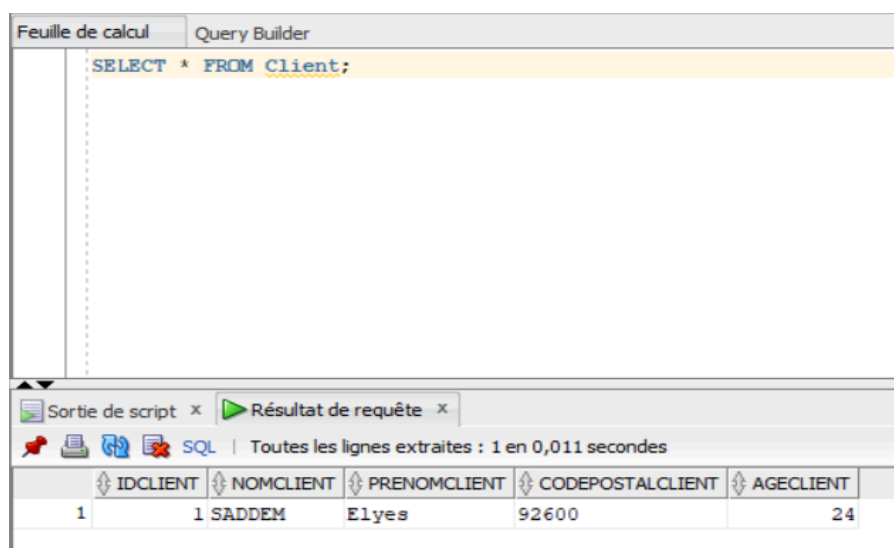
);

END;

```



On vérifie l'ajout de l'utilisateur.



Maintenant, on tente d'ajouter un utilisateur sans âge.


```
BEGIN

packageClient.addClient(

    p_idClient => 2,

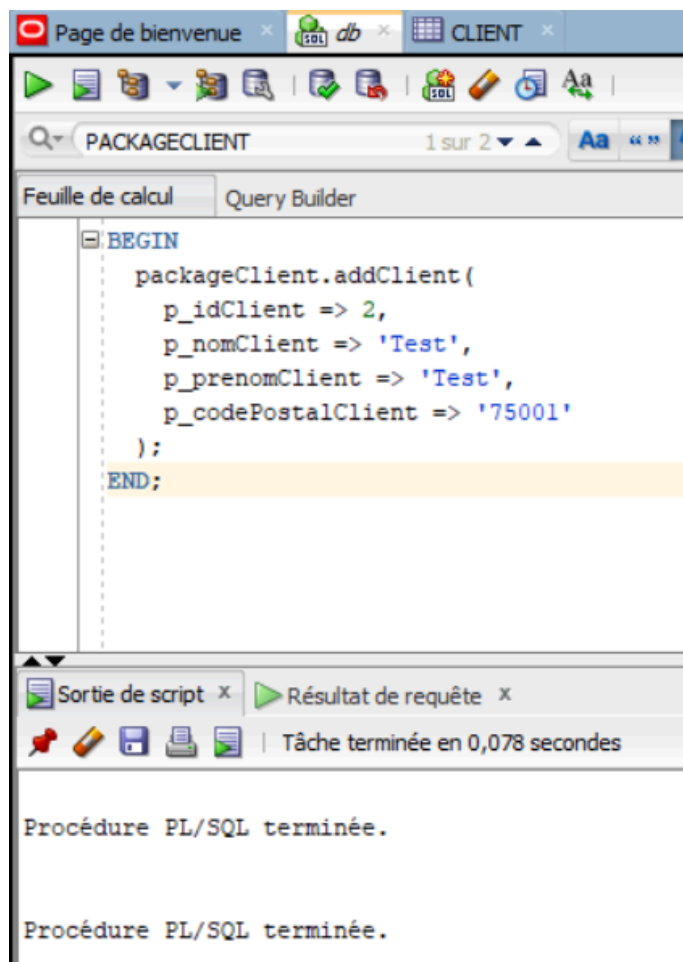
    p_nomClient => 'Test',

    p_prenomClient => 'Test',

    p_codePostalClient => '75001'

);

END;
```



On vérifie notamment l'ajout de cet utilisateur.

Feuille de calcul		Query Builder		
		SELECT * FROM Client;		

Sortie de script x		Résultat de requête x		
		SQL Toutes les lignes extraites : 2 en 0,003 secondes		
IDCLIENT	NOMCLIENT	PRENOMCLIENT	CODEPOSTALCLIENT	AGECLIENT
1	1 SADDEM	Elyes	92600	24
2	2 Test	Test	75001	(null)

Suite à cela, on va tenter d'ajouter un utilisateur ayant un identifiant existant:

```
BEGIN

packageClient.addClient(

    p_idClient => 1, -- ID déjà utilisé dans le test 1

    p_nomClient => 'Doe',

    p_prenomClient => 'John',

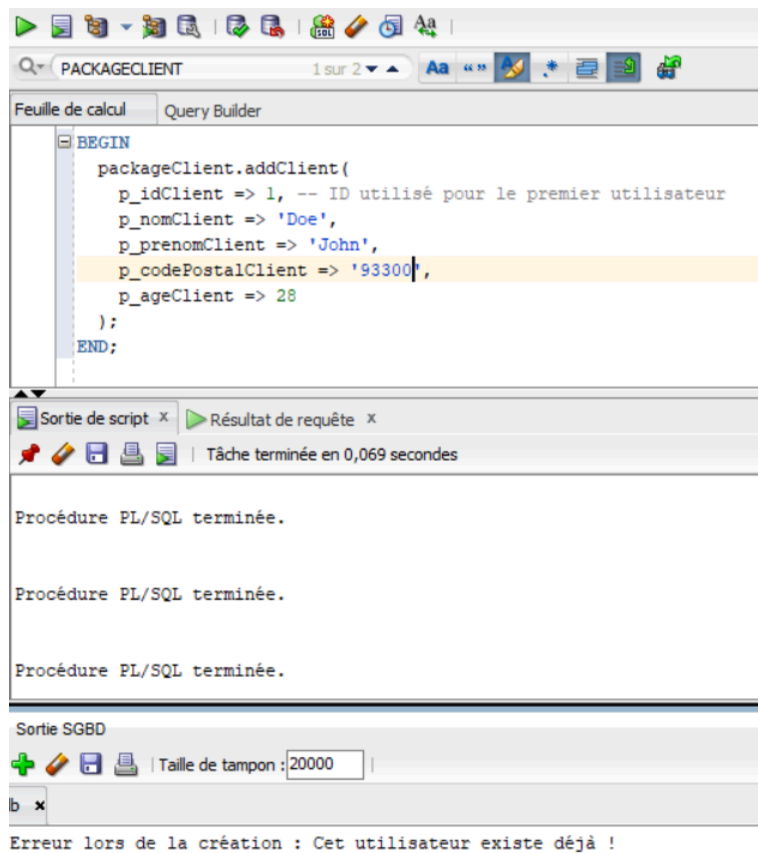
    p_codePostalClient => '93300',

    p_ageClient => 28

);

END;
```

Une exception sera levée.



Pour terminer, on va ajouter un autre utilisateur et calculer la moyenne des âges correspondantes au niveau de la table Client.

```
BEGIN

packageClient.addClient(

  p_idClient => 3,

  p_nomClient => 'Smith',

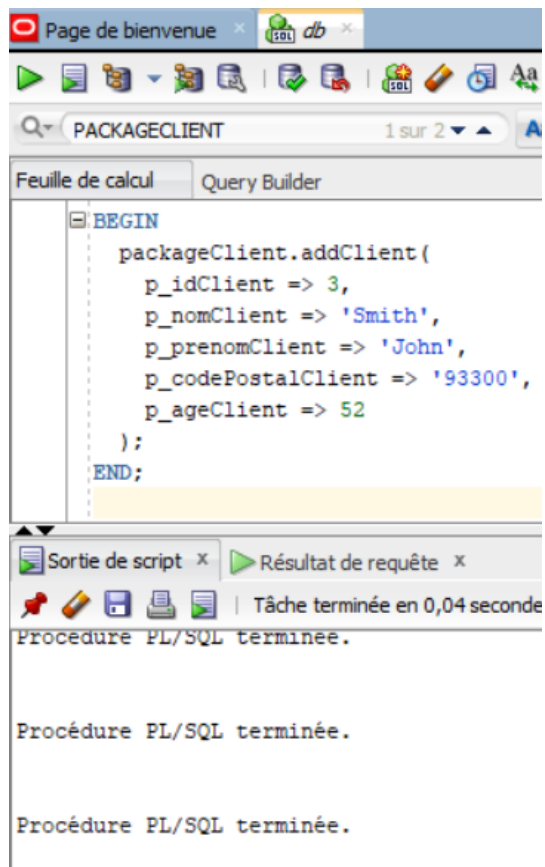
  p_prenomClient => 'John',

  p_codePostalClient => '93300',

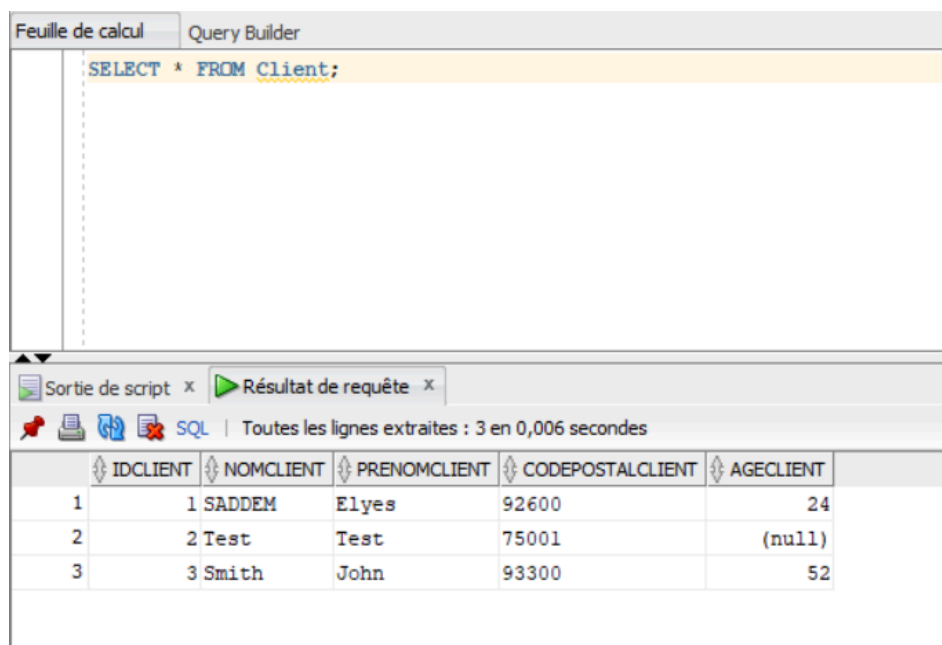
  p_ageClient => 52

);

END;
```



On vérifie l'ajout de l'utilisateur:



On procède maintenant au calcul de la moyenne :

```

DECLARE

    moyenne_age NUMBER;

```

```

BEGIN

    moyenne_age := packageClient.getAgeAvg;

    DBMS_OUTPUT.PUT_LINE('Âge moyen des clients : ' || moyenne_age);

END;

```

Le résultat obtenu est donc le suivant :

