



# Rapport pl/sql

2 ème année génie informatique

Encadré par : MR GHERABI NOUREDDINE

Réalisé par: **MHAMDI Jaouad**

**AIT OUCHAOUR Youssef**

## tp1:

### EXERCICE 1:

1. Créer un utilisateur portant votre nom et donner lui les privilèges DBA.

```
SQL> create user mhamdi identified by mhamdi;
User created.
SQL> grant sysdba to mhamdi;
Grant succeeded.
SQL>
```

```
SQL> grant all privileges to mhamdi;
Grant succeeded.
SQL>
```

2. Connectez-vous par l'utilisateur crée puis créer le schéma relationnel en dessus.

```
SQL> connect mhamdi/mhamdi;
Connected.
SQL>
```

```
SQL> create table AGENCE (
  2 Code_Agence Number(15) primary key,
  3 Nom varchar(15) not null,
  4 Ville varchar(15)
  5 );
Table created.
SQL> create table AGENT (
  2 Matricule varchar(15) primary key,
  3 Nom varchar(15),
  4 Prenom varchar(15),
  5 Salaire number(15),
  6 Date_Embauche date,
  7 Code_agence number(20),
  8 foreign key (Code_agence) REFERENCES Agence(Code_agence)
  9 );
Table created.
```

```

SQL> create table COMPTE (
  2  Num_Compte number(15) primary key,
  3  Code_Agence number(15),
  4  Matricule varchar(15),
  5  Solde number(15),
  6  check (Solde > 100),
  7  foreign key (Code_Agence) REFERENCES Agence(Code_Agence),
  8  foreign key (Matricule) REFERENCES AGENT(Matricule)
  9  );

```

Table created.

SQL>

```

SQL> desc agence

```

Name	Null?	Type
CODE_AGENCE	NOT NULL	NUMBER(15)
NOM	NOT NULL	VARCHAR2(15)
VILLE		VARCHAR2(15)

```

SQL> desc agent

```

Name	Null?	Type
MATRICULE	NOT NULL	VARCHAR2(15)
NOM		VARCHAR2(15)
PRENOM		VARCHAR2(15)
SALAIRE		NUMBER(15)
DATE_EMBAUCHE		DATE
CODE_AGENCE		NUMBER(20)

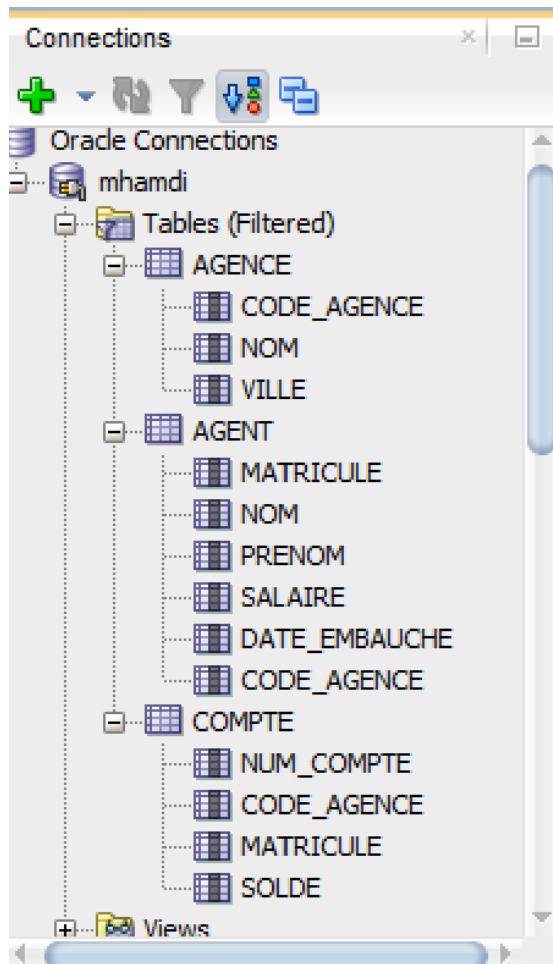
```

SQL> desc compte

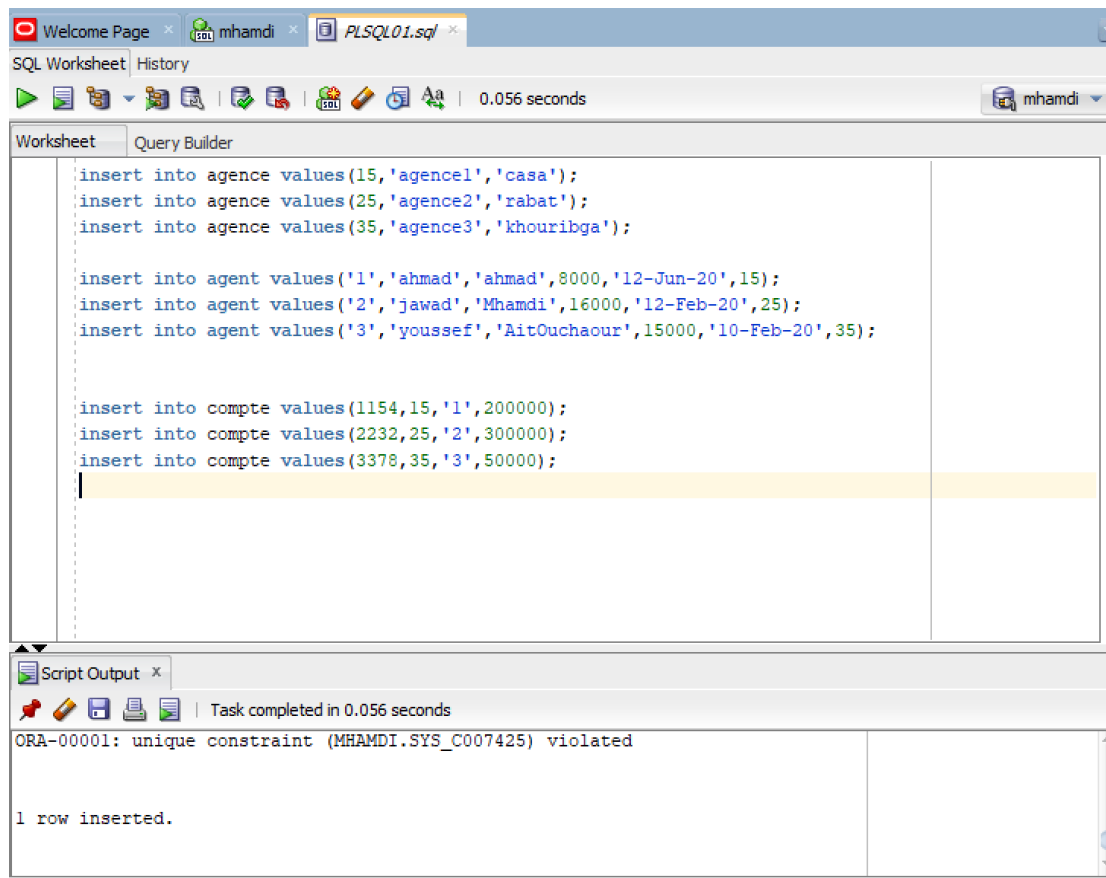
```

Name	Null?	Type
NUM_COMPTE	NOT NULL	NUMBER(15)
CODE_AGENCE		NUMBER(15)
MATRICULE		VARCHAR2(15)
SOLDE		NUMBER(15)

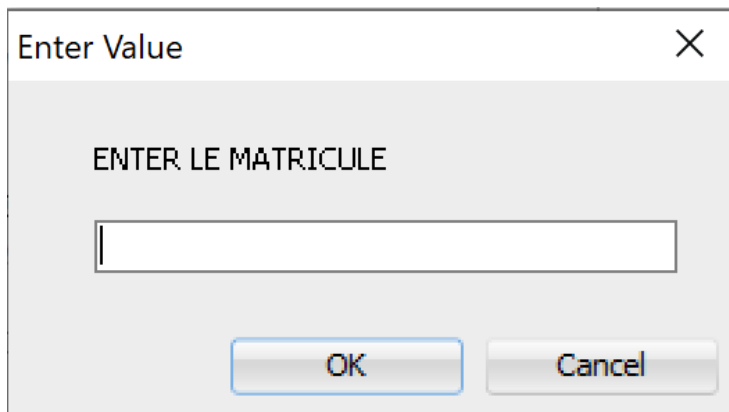
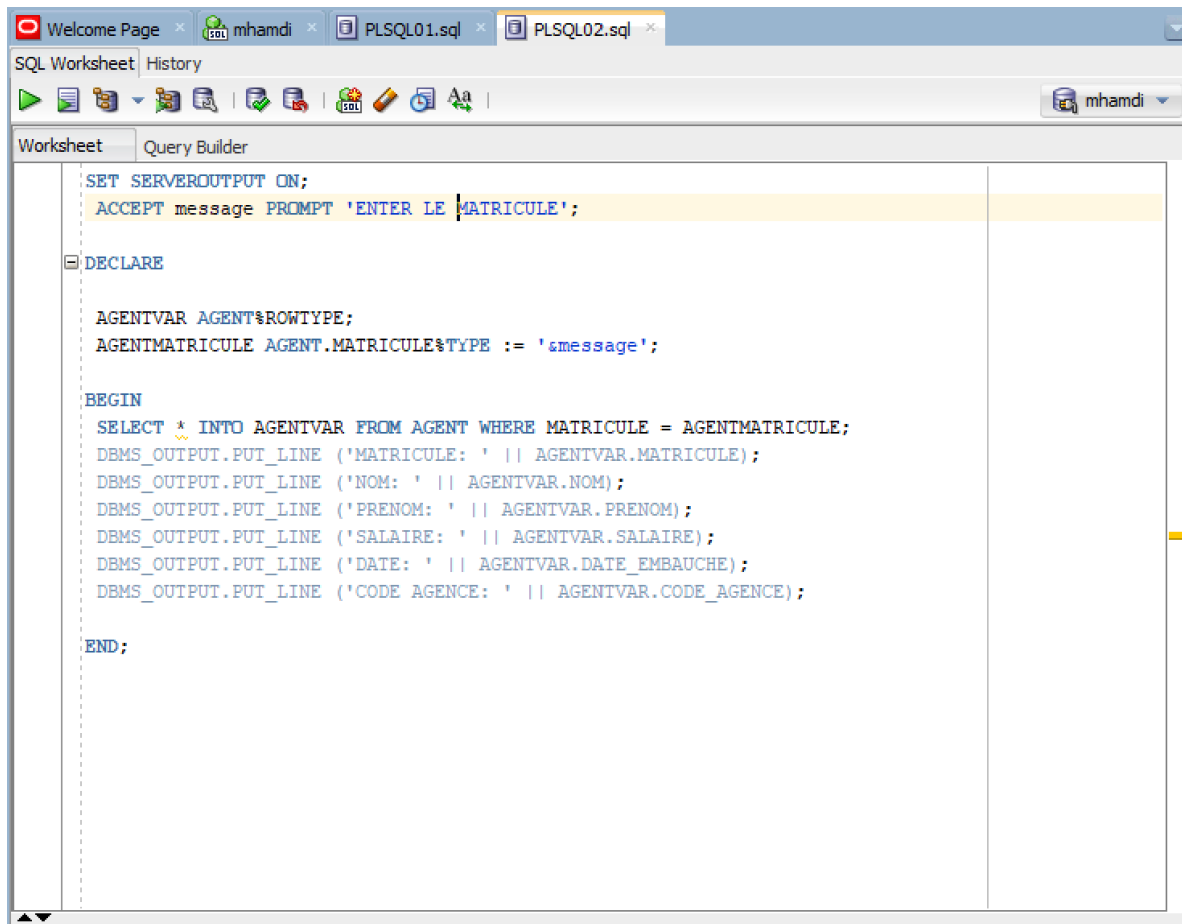
SQL>



3. PLSQL01: Insérer des jeux d'essai dans toutes les tables.



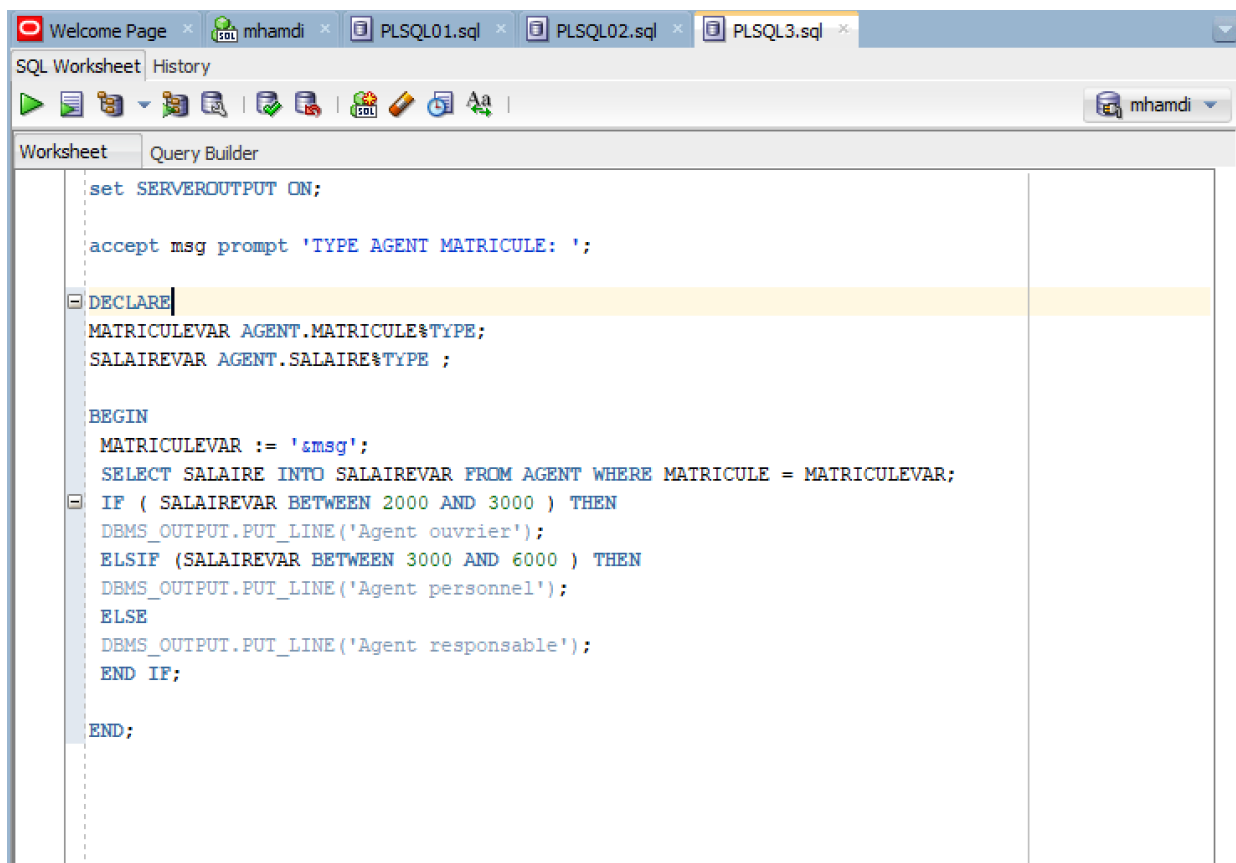
4. PLSQL02: Afficher les informations de l'agent dont la matricule est entré par l'utilisateur



```
END;
MATRICULE: 2
NOM: jawad
PRENOM: Mhamdi
SALAIRE: 16000
DATE: 12-02-0020
CODE AGENCE: 25
```

PL/SQL procedure successfully completed.

5. PLSQL03: à l'aide de l'instruction conditionnelle (IF ELSE THEN), afficher l'état de l'agent selon son salaire (le numéro est entré par l'utilisateur)



```
set SERVEROUTPUT ON;

accept msg prompt 'TYPE AGENT MATRICULE: ';

DECLARE
MATRICULEVAR AGENT.MATRICULE%TYPE;
SALAIREVAR AGENT.SALAIRE%TYPE ;

BEGIN
MATRICULEVAR := '&msg';
SELECT SALAIRE INTO SALAIREVAR FROM AGENT WHERE MATRICULE = MATRICULEVAR;
IF ( SALAIREVAR BETWEEN 2000 AND 3000 ) THEN
DBMS_OUTPUT.PUT_LINE('Agent ouvrier');
ELSIF (SALAIREVAR BETWEEN 3000 AND 6000 ) THEN
DBMS_OUTPUT.PUT_LINE('Agent personnel');
ELSE
DBMS_OUTPUT.PUT_LINE('Agent responsable');
END IF;

END;
```

```

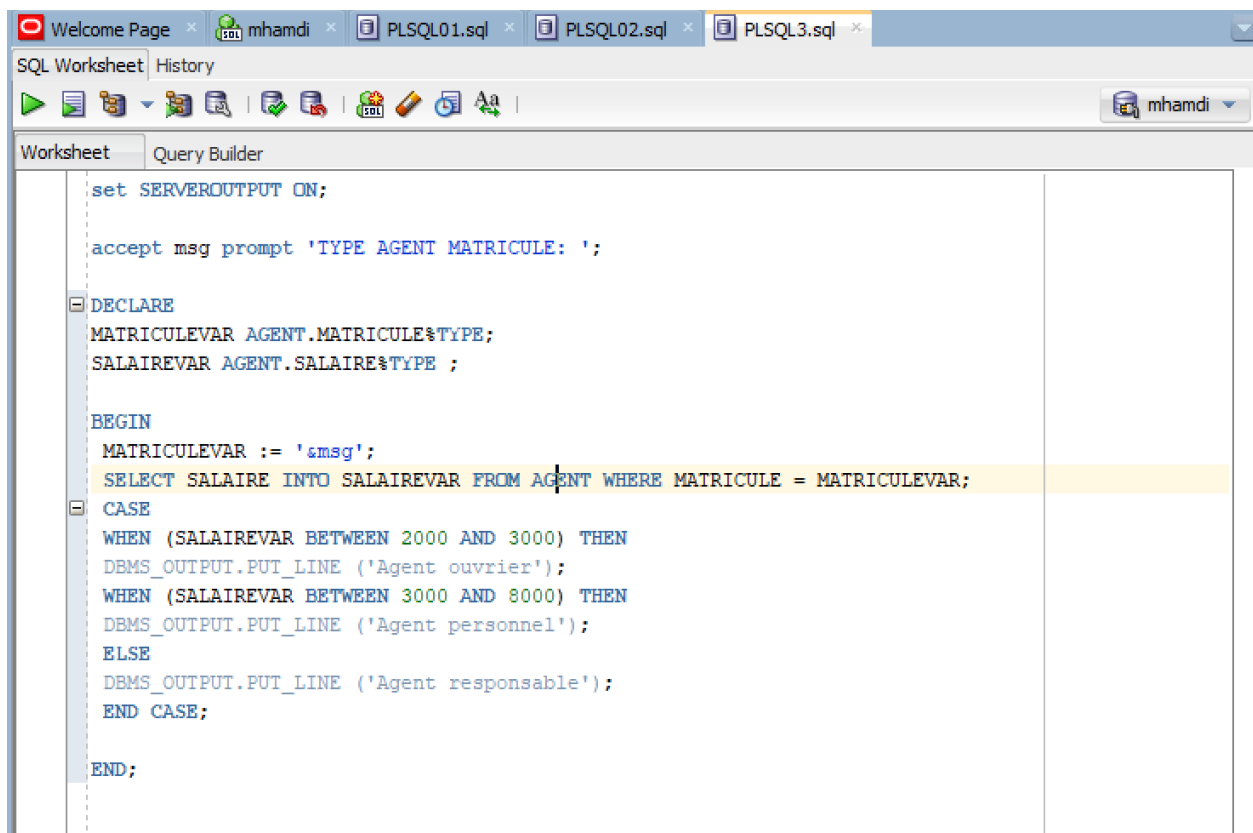
SELECT SALAIRE INTO SALAIREVAR FROM AGENT WHERE MATRICULE = MATRICULEVAR;
IF ( SALAIREVAR BETWEEN 2000 AND 3000 ) THEN
DBMS_OUTPUT.PUT_LINE('Agent ouvrier');
ELSIF (SALAIREVAR BETWEEN 3000 AND 6000 ) THEN
DBMS_OUTPUT.PUT_LINE('Agent personnel');
ELSE
DBMS_OUTPUT.PUT_LINE('Agent responsable');
END IF;

END;
Agent responsable

PL/SQL procedure successfully completed.

```

## 6. PLSQL03: Refaire la question 5 avec l'instruction CASE



The screenshot shows an SQL Worksheet window with the following content:

```

Welcome Page x mhamdi x PLSQL01.sql x PLSQL02.sql x PLSQL3.sql x
SQL Worksheet History
set SERVEROUTPUT ON;

accept msg prompt 'TYPE AGENT MATRICULE: ';

DECLARE
MATRICULEVAR AGENT.MATRICULE%TYPE;
SALAIREVAR AGENT.SALAIRE%TYPE ;

BEGIN
MATRICULEVAR := '&msg';
SELECT SALAIRE INTO SALAIREVAR FROM AGENT WHERE MATRICULE = MATRICULEVAR;

CASE
WHEN (SALAIREVAR BETWEEN 2000 AND 3000) THEN
DBMS_OUTPUT.PUT_LINE ('Agent ouvrier');
WHEN (SALAIREVAR BETWEEN 3000 AND 8000) THEN
DBMS_OUTPUT.PUT_LINE ('Agent personnel');
ELSE
DBMS_OUTPUT.PUT_LINE ('Agent responsable');
END CASE;

END;

```



```

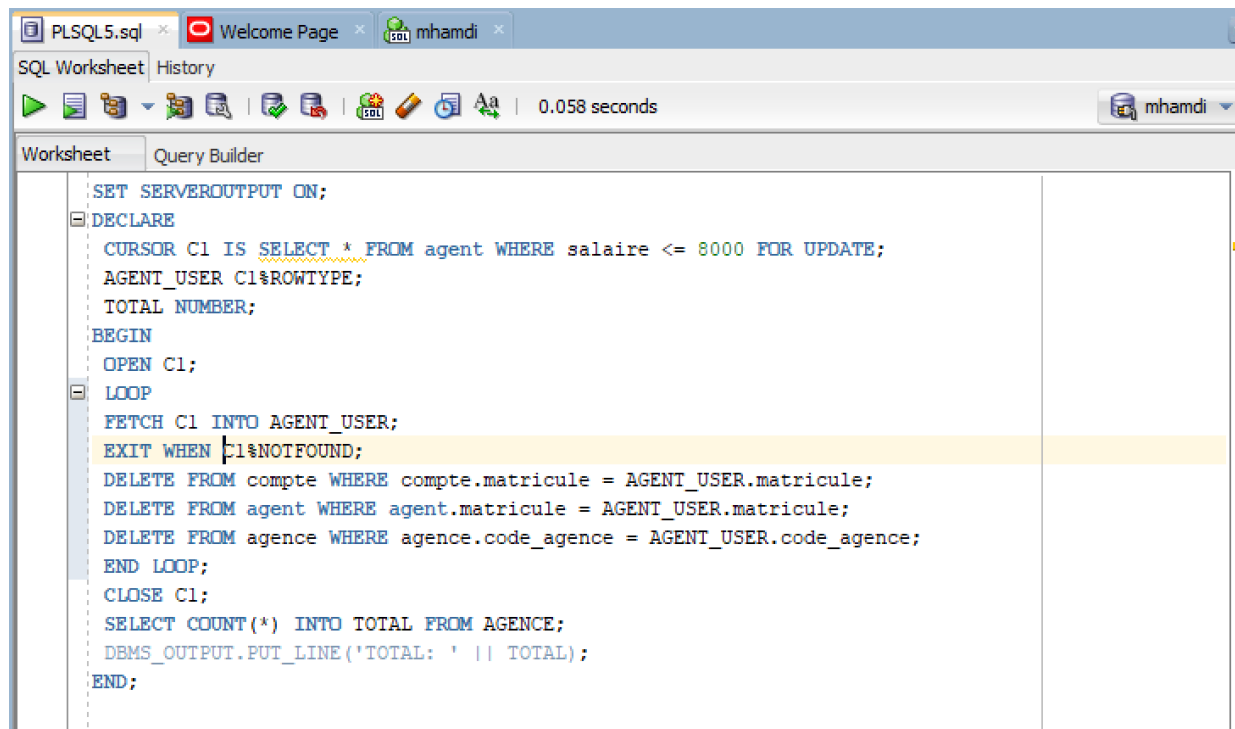
BEGIN
  MATRICULEVAR := '2';
  SELECT SALAIRE INTO SALAIREVAR FROM AGENT WHERE MATRICULE = MATRICULEVAR;
  CASE
  WHEN (SALAIREVAR BETWEEN 2000 AND 3000) THEN
    DBMS_OUTPUT.PUT_LINE ('Agent ouvrier');
  WHEN (SALAIREVAR BETWEEN 3000 AND 8000) THEN
    DBMS_OUTPUT.PUT_LINE ('Agent personnel');
  ELSE
    DBMS_OUTPUT.PUT_LINE ('Agent responsable');
  END CASE;

END;
Agent responsable

PL/SQL procedure successfully completed.

```

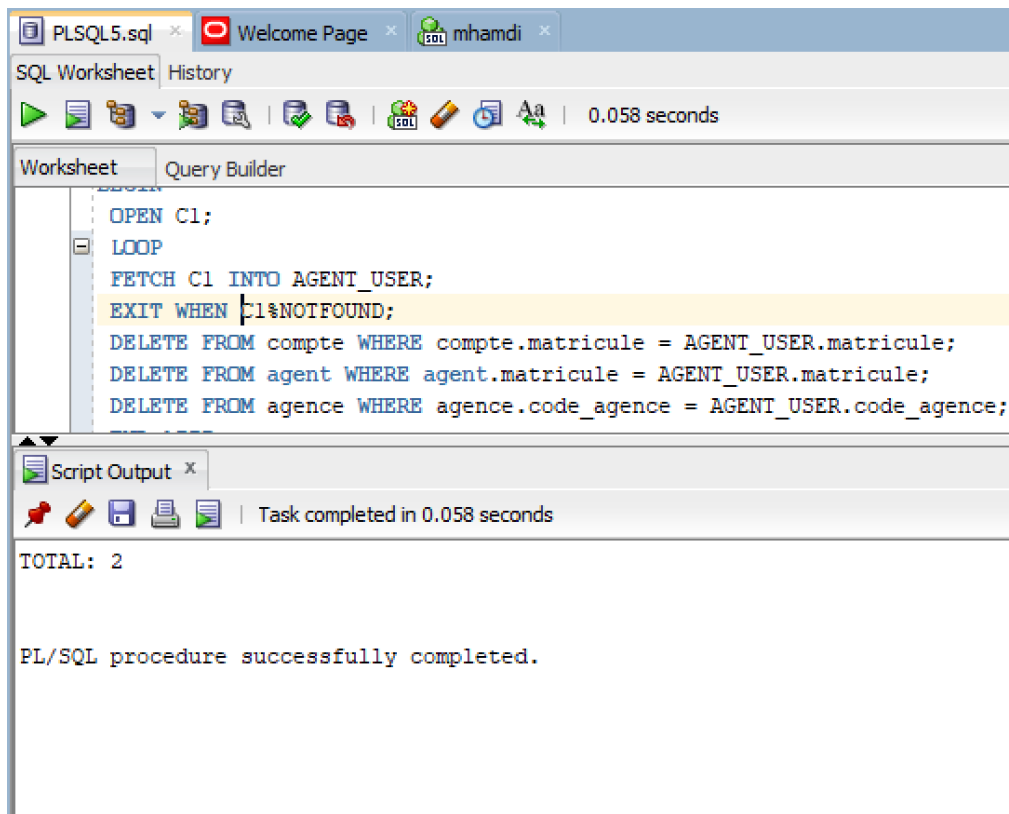
7. PLSQL05: En utilisant les curseurs, supprimer les agences gérées par les agents dont leur salaire est inférieur à 8000dh.



```

SET SERVEROUTPUT ON;
DECLARE
  CURSOR C1 IS SELECT * FROM agent WHERE salaire <= 8000 FOR UPDATE;
  AGENT_USER C1%ROWTYPE;
  TOTAL NUMBER;
BEGIN
  OPEN C1;
  LOOP
    FETCH C1 INTO AGENT_USER;
    EXIT WHEN C1%NOTFOUND;
    DELETE FROM compte WHERE compte.matricule = AGENT_USER.matricule;
    DELETE FROM agent WHERE agent.matricule = AGENT_USER.matricule;
    DELETE FROM agence WHERE agence.code_agence = AGENT_USER.code_agence;
  END LOOP;
  CLOSE C1;
  SELECT COUNT(*) INTO TOTAL FROM AGENCE;
  DBMS_OUTPUT.PUT_LINE('TOTAL: ' || TOTAL);
END;

```



8. Créer deux tables PR1 (Num\_Compte Number) et PR2 (Num\_Compte Number) contenant un seul champ Num\_Compte.

```

SQL> create table TB1(
  2  num_compte number(20)
  3  );

Table created.

SQL> create table TB2(
  2  num_compte number(20)
  3  );

Table created.

SQL>

```

un seul champ Num\_Compte. 9. PLSQL06: En utilisant les curseurs, insérer les comptes ayant un solde inférieur à 10000dh dans la table PR1 et les comptes ayant un solde supérieur à 10000dh dans la table PR2.

PLSQL5.sql x Welcome Page x mhamdi x PLSQL06.sql x

SQL Worksheet History

Worksheet Query Builder

```

SET SERVEROUTPUT ON;
DECLARE
    CURSOR C1 IS SELECT num_compte, solde FROM compte ;
    num_compte compte.num_compte%TYPE;
    solde compte.solde%TYPE;
BEGIN
    OPEN C1;
    LOOP
        FETCH C1 INTO num_compte, solde;
        EXIT WHEN C1%NOTFOUND;
        IF ( solde < 10000 ) THEN
            INSERT INTO TB1 VALUES ( num_compte );
        ELSE
            INSERT INTO TB2 VALUES ( num_compte );
        END IF;
    END LOOP;
END;

```

PLSQL06.sql x TB1 x PLSQL5.sql x Welcome Page x mhamdi x

Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details

Sort.. Filter:

	NUM_COMPTE
1	7654
2	9876
3	3476

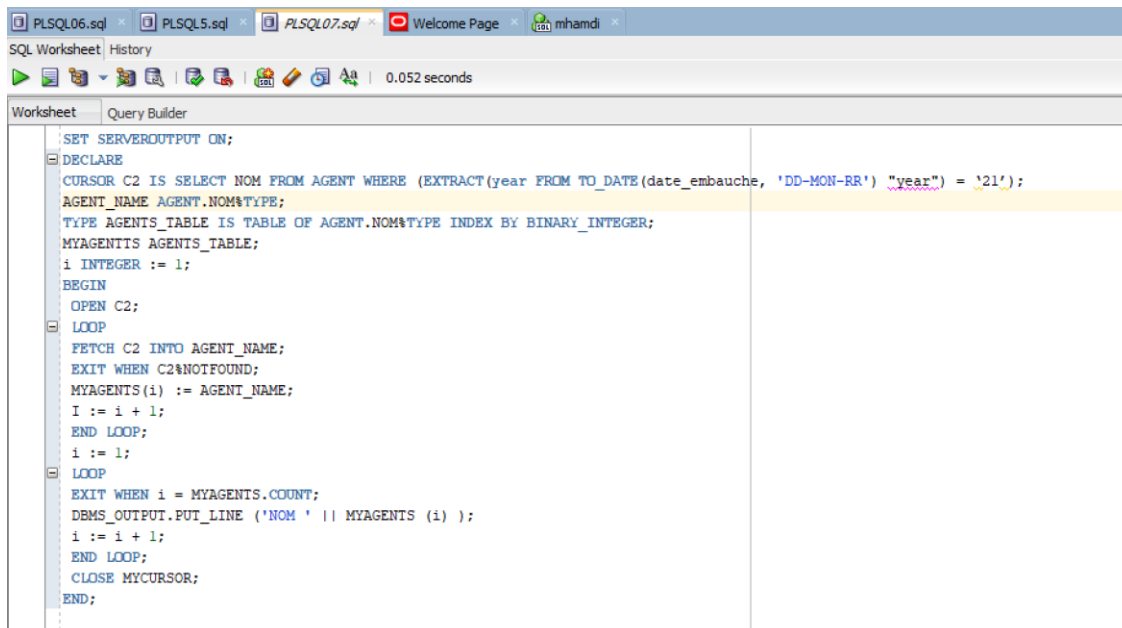
PLSQL06.sql x TB2 x PLSQL5.sql x Welcome Page x mhamdi x

Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Index

Sort.. Filter:

	NUM_COMPTE
1	2232
2	3378
3	1265
4	2232
5	3378

10. PLSQL07: Insérer les noms des agents embauchés en 2021 dans un tableau indexé par des entiers, puis afficher le contenu du tableau.



```
SET SERVEROUTPUT ON;
DECLARE
CURSOR C2 IS SELECT NOM FROM AGENT WHERE (EXTRACT(year FROM TO_DATE(date_embauche, 'DD-MON-RR') "year") = '21');
AGENT_NAME AGENT.NOM%TYPE;
TYPE AGENTS_TABLE IS TABLE OF AGENT.NOM%TYPE INDEX BY BINARY_INTEGER;
MYAGENTS AGENTS_TABLE;
i INTEGER := 1;
BEGIN
OPEN C2;
LOOP
FETCH C2 INTO AGENT_NAME;
EXIT WHEN C2%NOTFOUND;
MYAGENTS(i) := AGENT_NAME;
i := i + 1;
END LOOP;
i := 1;
LOOP
EXIT WHEN i = MYAGENTS.COUNT;
DBMS_OUTPUT.PUT_LINE ('NOM ' || MYAGENTS (i) );
i := i + 1;
END LOOP;
CLOSE MYCURSOR;
END;
```

PLSQL08: Écrire un bloc PL/SQL permettant de transférer les données de la table «Agence» dans la table « Agence1» à savoir:

- Si la table « Agence » est vide, la table « Agence1» devra contenir uniquement le n-uplet (0,'Aucune Agence, 'Aucune ville'), sinon accéder séquentiellement à l'aide d'un curseur et un enregistrement, effectuer les transformations sur les champs et stocker le résultat dans la table « Agence1» tel que:

- Le Nom est écrit en majuscule,
- Le Code Agence = Code Agence + 1

```
SET SERVEROUTPUT ON;

DECLARE
CURSOR C3 IS SELECT * FROM agence;
agenc AGENCE%ROWTYPE;
TOTAL_AGENCES NUMBER;

BEGIN
SELECT COUNT (*) INTO TOTAL_AGENCES FROM AGENCE;
OPEN C3;
IF TOTAL_AGENCES != 0 THEN
LOOP
FETCH C3 INTO agenc;
EXIT WHEN C3%NOTFOUND;
INSERT INTO AGENCE1 VALUES (
agenc.NUM_AGENCE + 1,
UPPER(agenc.NOM),
agenc.VILLE
);
END LOOP;
ELSE
INSERT INTO AGENCE1 VALUES (
0,
'AUCUNE AGENCE',
'AUCUNE VILLE'
);
END IF;
CLOSE C3;
END;
```

tp2:

Soit le schéma relationnel suivant:

**Employe (Num\_Enmp, Nom, prénom, age, ville, salaire, #Num\_ Service)**

**Service (Num\_Service, nom\_service, departement)**

1. Créer un utilisateur nommé «Emp\_user» et donner lui les privilèges DBA

```
SQL> create user Emp_user identified by mhamdi;
User created.
SQL> grant all privileges to Emp_user;
Grant succeeded.
SQL>
```

2. Connectez-vous par l'utilisateur créé puis créer le schéma relationnel en dessus:

```
SQL> create table service(
  2  num_service Number(15) primary key,
  3  nom_service varchar(15),
  4  departement varchar(15)
  5  );
Table created.
SQL> create table employe(
  2  num_emp Number(15) primary key,
  3  nom varchar(15),
  4  prenom varchar(15),
  5  age Number(15),
  6  ville varchar(15),
  7  salaire Number(15),
  8  num_service Number(15),
  9  foreign key (num_service) references service(num_service)
 10 );
Table created.
SQL>
```

3 Créez une procédure nommée « ADD-Service» qui permet d'insérer les informations d'un service dans la table « Service ». les informations du service sont indiquées dans les paramètres de la procédure:

The screenshot shows the SQL Developer interface with the 'Query Builder' tab active. The SQL editor contains the following code:

```

CREATE OR REPLACE PROCEDURE ADD_service
(num_servicel IN service.num_service%TYPE,
nom_servicel IN service.nom_service%TYPE,
departementl IN service.departement%TYPE)
IS
BEGIN
    INSERT INTO service VALUES (num_servicel,nom_servicel,departementl);
END;
/

```

4. Compilez et testez la procédure « ADD\_Service » en ajoutant 3 services:

```

BEGIN
ADD_service(1,'service1','dep1');
ADD_service(2,'service2','dep2');
ADD_service(3,'service3','dep2');
END;

```

The screenshot shows the SQL Developer interface with the 'SERVICE' table selected. The 'Data' tab is active, displaying the following data:

	NUM_SERVICE	NOM_SERVICE	DEPARTEMENT
1	1	service1	dep1
2	2	service2	dep2
3	3	service3	dep2

5. On veut créer une procédure qui permet d'insérer un nouvel employé dans la table «Employe ».

La procédure doit contenir un appel à la fonction « VALID\_NoService » qui vérifie si le numéro de service indiqué pour le nouvel employé existe dans la table « Service ».

5.1 Créez la fonction « VALID NoService » pour valider un numéro de service indiqué. La fonction doit renvoyer une valeur BOOLEAN (True si le numéro existe et False si une exception est levée concernant le numéro inexistant)

```
create or replace FUNCTION valid_NoService
(num_s in service.num_service%TYPE)
RETURN boolean
IS
    nbr NUMBER;
BEGIN
    select count(*) into nbr from service where num_service=num_s;
    return nbr>0;
END valid_NoService;
```

5.2 Créez la procédure «ADD\_Employe » pour insérer un employé dans la table «Employe». l'employé est ajouté à la table si la fonction renvoie TRUE, sinon une exception est levée qui affiche une alerte à l'utilisateur.

```
create or replace PROCEDURE ADD_Employe
(numero in employee.num_Emp%type, nom in employee.nom%type, prenom in employee.prenom%type ,
age in employee.age%type,
ville in employee.ville%type, salaire in employee.salaire%type,
Num_Service in service.num_service%TYPE)
IS
    e EXCEPTION;
BEGIN
    if valid_noservice(num_service)=FALSE then
        raise e;
    else
        dbms_output.put_line('num service existe');
        insert into employee VALUES(numero,nom,prenom,age,ville,salaire,num_service);
        commit;
    end if;
Exception
    when e then
        dbms_output.put_line('num service existe pas');
END;
```

6. Créez une procédure qui affiche les enregistrements d'un employé (son code est entré par l'utilisateur), vérifier également le traitement des exceptions en tentant d'afficher des données inexistantes ou tenter de se connecter avec un utilisateur inexistant.



Worksheet Query Builder

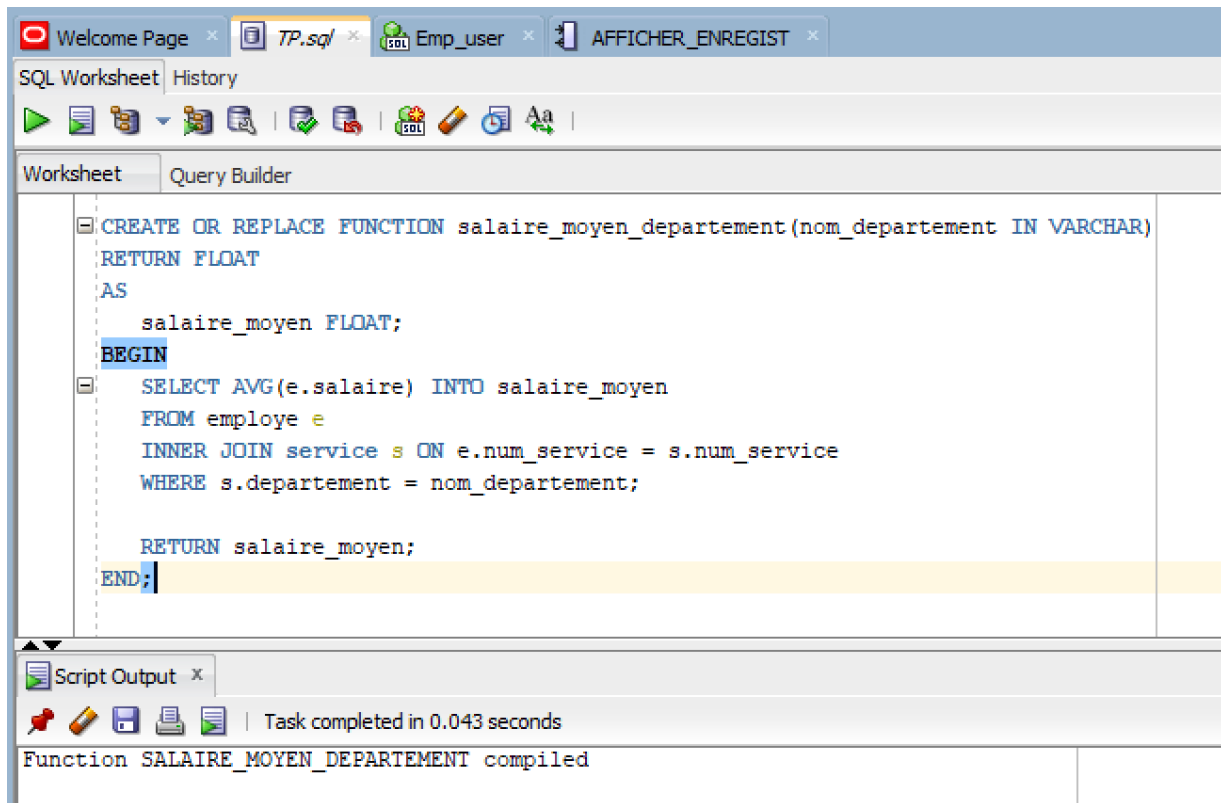
```
create or replace NONEDITIONABLE PROCEDURE afficher_enregist(code_employe IN INT)
AS
    nom_employe VARCHAR(15);
    prenom_employe VARCHAR(15);
BEGIN
    BEGIN
        SELECT nom, prenom INTO nom_employe, prenom_employe
        FROM employe
        WHERE employe.num_emp = code_employe;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('L'employé avec le code ' || code_employe || ' n'existe pas.');
```

Script Output x

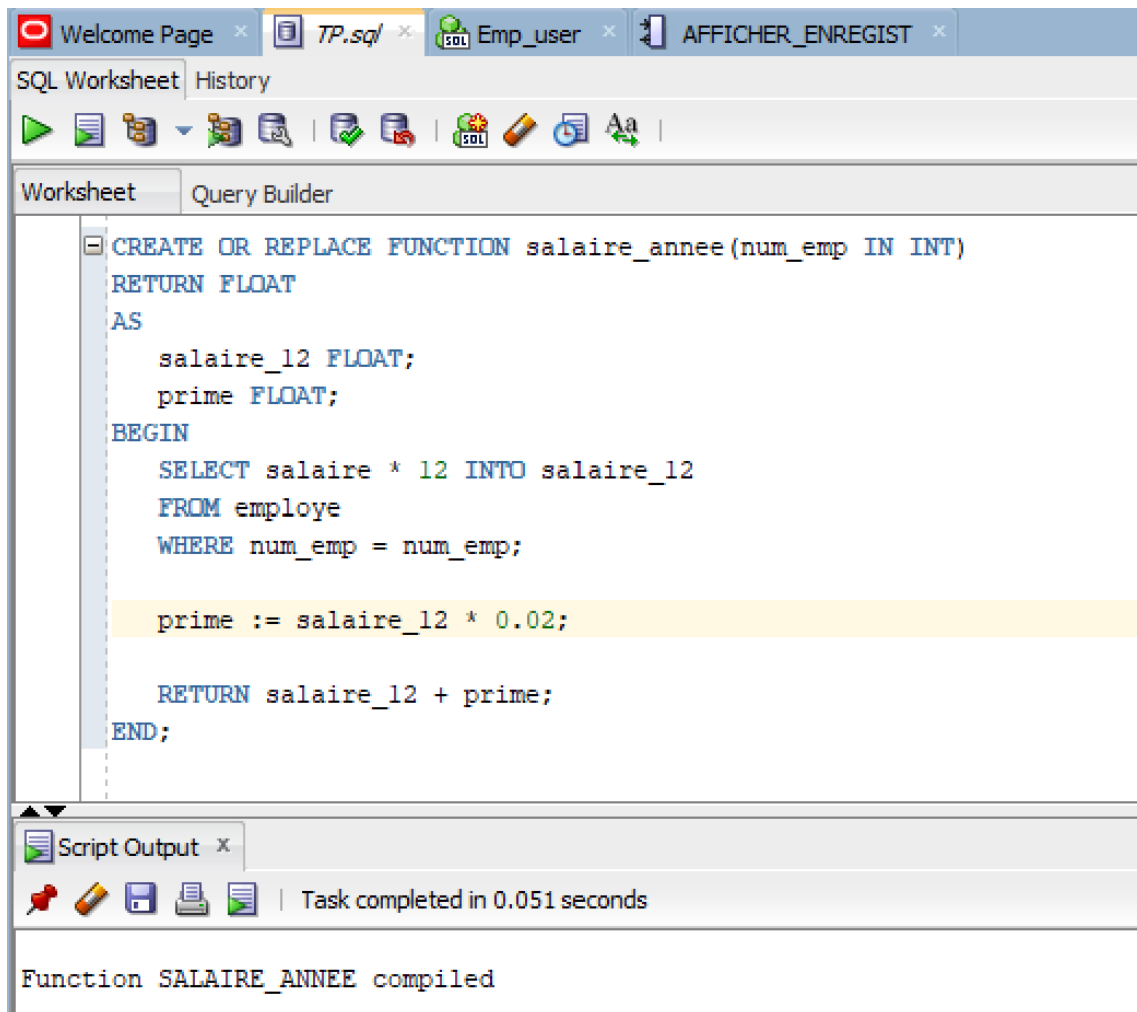
Task completed in 0.045 seconds

Procedure AFFICHER\_ENREGIST compiled

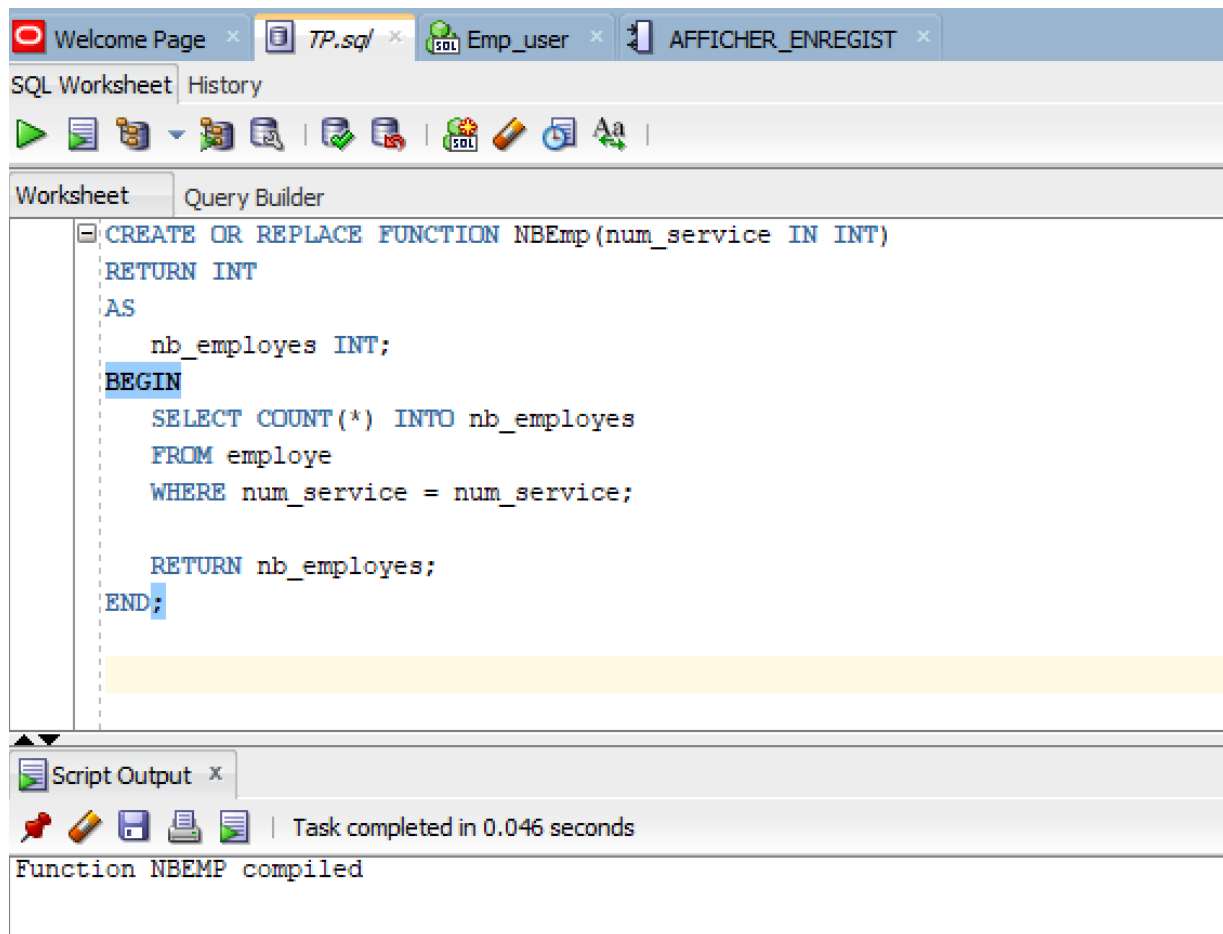
7. Ecrivez une fonction PL/SQL permettant de retourner le salaire moyen de tous les employés d'un département donné.



8. Ecrivez une fonction qui renvoie le salaire de l'année d'un employé donné. Le salaire de l'année est défini par la formule: (salaire 12) + Prime, Tel que "Prime" représente 2% du salaire annuel (salaire\*12)



9. Ecrivez une fonction « NBEmp » qui renvoie le nombre des employés qui travaillent dans le service numéro 3 (le numéro de service est entré comme paramètre).



### tp3:

Soit le modèle relationnel suivant:

**EMPLOYE (Matr, NomE, Poste, DatEmb, Age, Salaire (>1000))**

**PROJET (CodeP, NomP)**

**PARTICIPATION (#Matr, # CodeP)**

1. Créer un utilisateur portant votre nom et donner lui les privilèges DBA.

```
SQL> create user jaouad identified by mhamdi;
User created.

SQL> grant all privileges to jaouad;
Grant succeeded.

SQL>
```

2. Connectez-vous par l'utilisateur crée puis créer le schéma relationnel en dessus:

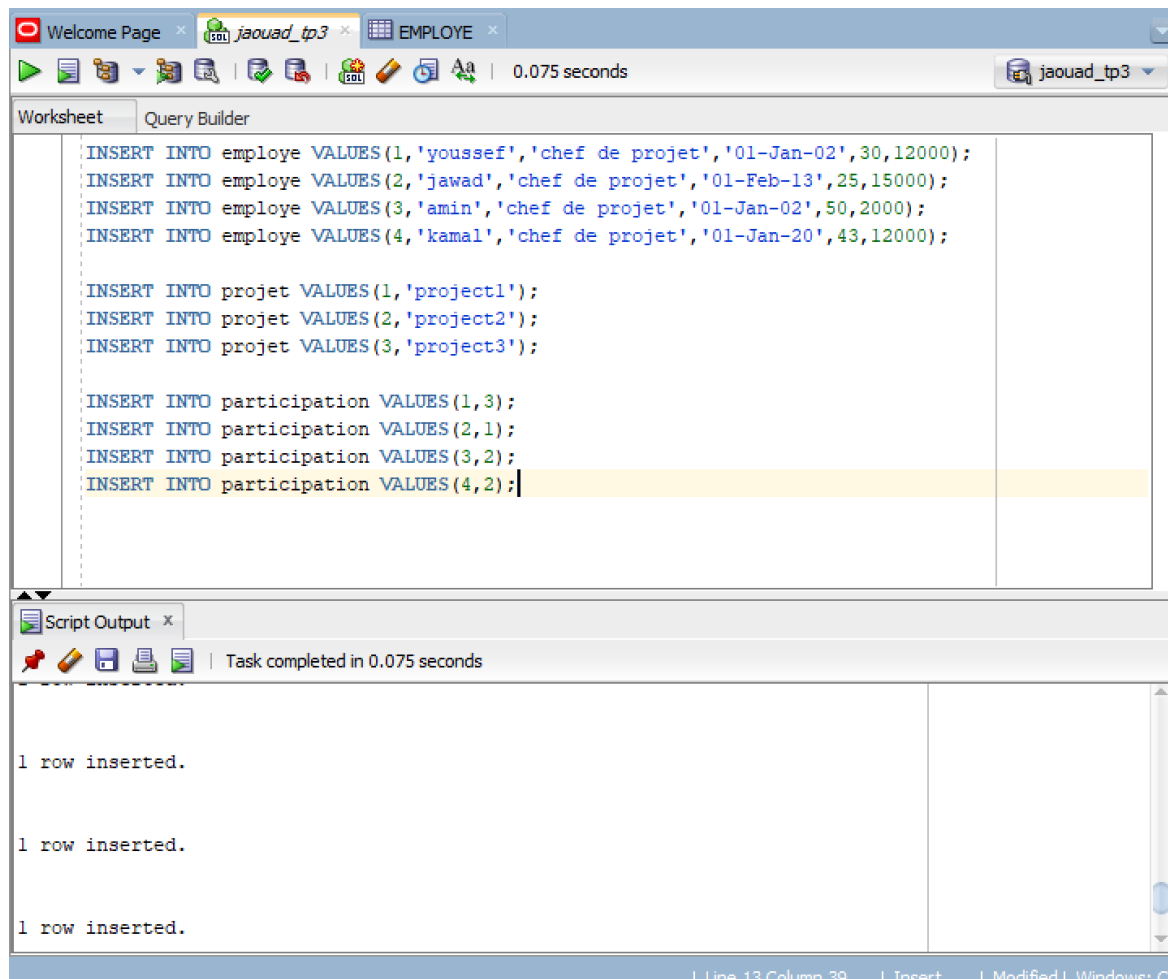
```
SQL> connect jaouad/mhamdi;
Connected.
SQL> create table employe(
  2  matr varchar(20) primary key,
  3  nome varchar(20),
  4  Poste varchar(20),
  5  DatEmb date,
  6  Age Number,
  7  Salaire Number(7),
  8  check (Salaire > 1000));
Table created.
```

```
SQL> create table PROJET(
  2  CodeP varchar(20) primary key,
  3  NomP varchar(20)
  4  );
Table created.

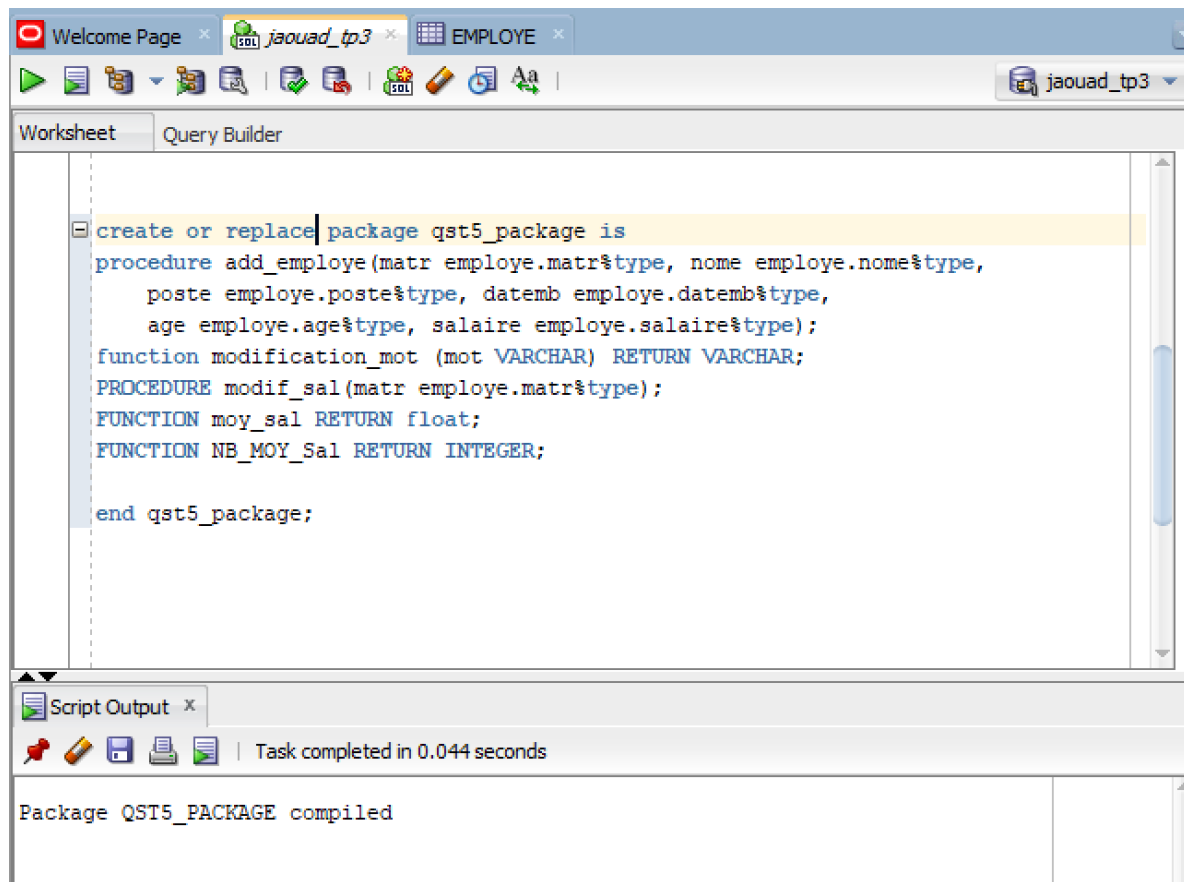
SQL> create table PARTICIPATION(
  2  matr varchar(20),
  3  CodeP varchar(20),
  4  foreign key (matr) references employe (matr),
  5  foreign key (CodeP) references PROJET (CodeP)
  6  );
Table created.

SQL>
```

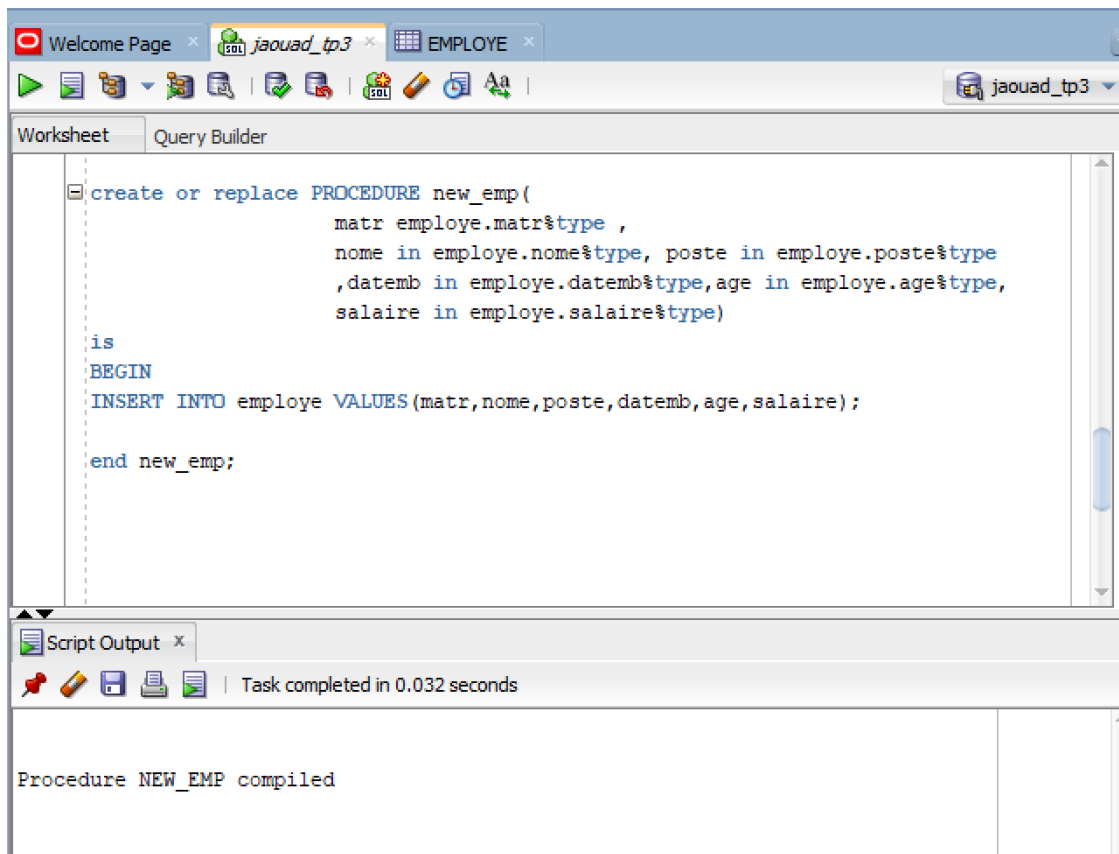
3. Ecrire un programme PL/SQL qui insère des données dans les différentes tables:



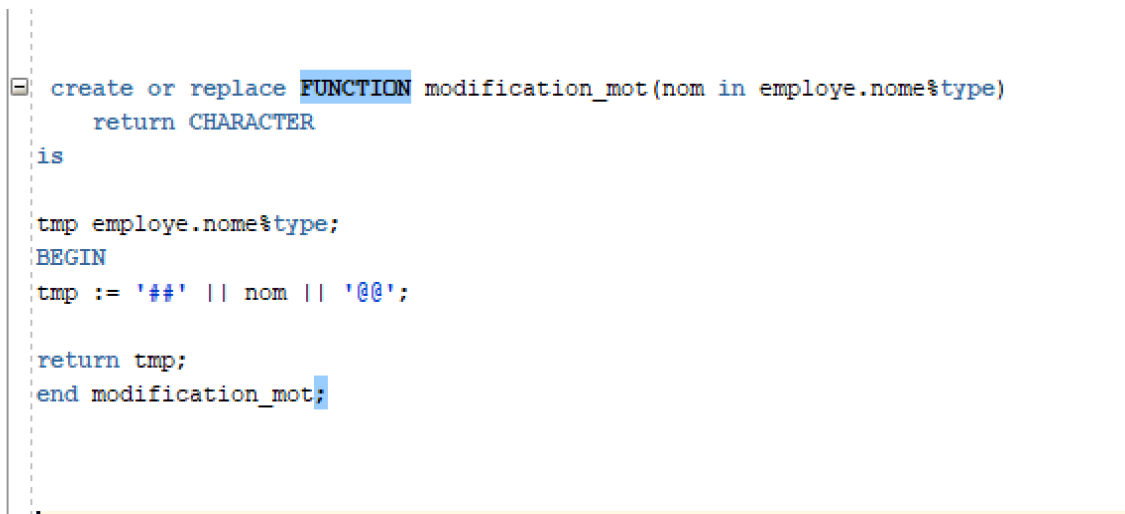
4. Ecrire une spécification package qui contient les procédures et les fonctions suivantes:



- Une procédure permettant d'insérer un nouveau Employé dont la matricule est entrée par l'utilisateur.



- Ecrire une fonction « modification\_mot» qui retourne un mot donné sous cette forme ##Mot@@» par exemple « ##Ahmadi@@»



- Une procédure permettant de modifier le salaire des employés selon leurs postes
  - Si l'employé est un technicien son salaire est augmenté par une prime de 500dh
  - Si l'employé est un ingénieur son salaire est augmenté par une prime de 1500dh



- Si l'employé est un manager son salaire est augmenté par une prime de 2500dh
- la procédure doit gérer une exception concernant les données inexistantes
- la procédure gère aussi exception dans le cas d'un curseur non valide

```

SET SERVEROUTPUT ON;
create or replace PROCEDURE mod_salaire(
    matr in employee.matr%type)
is
CURSOR c1 is select * from employee;
v_employe employee%rowtype;
BEGIN
    open c1;
    loop
        FETCH c1 into v_employe;
        EXIT when c1%notfound;
        case
            when(v_employe.poste = 'technicien') then
                update employee set salaire = v_employe.salaire + 500
                where matr = v_employe.matr;
            when(v_employe.poste = 'ingenieur') then
                update employee set salaire = v_employe.salaire + 1500
                where matr = v_employe.matr;
            when(v_employe.poste = 'manager') then
                update employee set salaire = v_employe.salaire + 500
                where matr = v_employe.matr;
            end case;
        end loop;
        close c1;
    exception
        when no_data_found then
            DBMS_OUTPUT.put_line ('no_data_found');
        when invalid_cursor then
            DBMS_OUTPUT.put_line ('non valide');
    end;

```

```

BEGIN
mod_salaire(6);
END;

```

The screenshot shows a database management interface with a tab labeled 'EMPLOYEE'. Below the tab is a menu bar with options: Columns, Data, Model, Constraints, Grants, Statistics, Triggers, Flashback, Dependencies, and De. Below the menu bar is a toolbar with icons for various database operations. The main area displays a table with the following columns: MATR, NOME, POSTE, DATEMB, AGE, and SALAIRE. The table contains five rows of data.

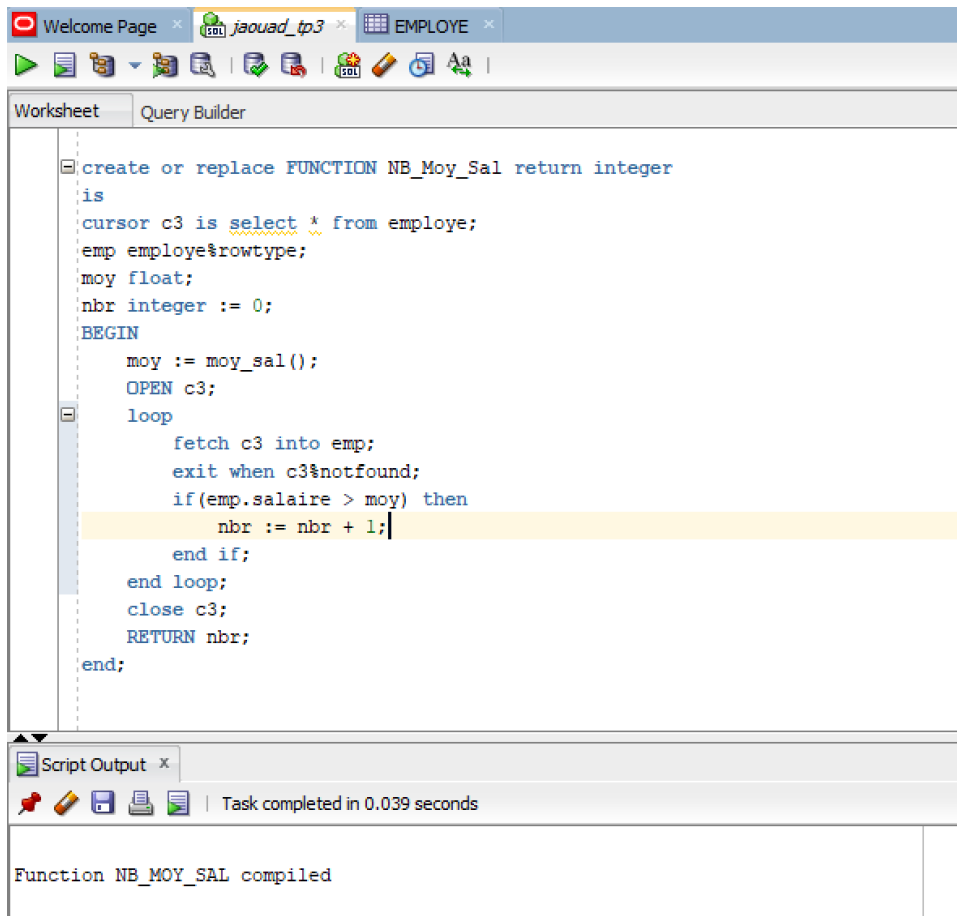
	MATR	NOME	POSTE	DATEMB	AGE	SALAIRE
1	1	youssef	chef de projet	01-JAN-02	30	12000
2	2	jawad	chef de projet	01-FEB-13	25	15000
3	3	amin	chef de projet	01-JAN-02	50	2000
4	4	kamal	chef de projet	01-JAN-20	43	12000
5	6	jawad	ingenieur	01-JAN-12	33	12000

- Une fonction « Moy\_Sal » permettant de calculer la moyenne des salaires pour les employés ayant un âge supérieur à 20 ans.

The screenshot shows a SQL query editor window with the following tabs: 'Welcome Page', 'jaouad\_tp3', and 'EMPLOYEE'. The 'Query Builder' tab is active. The editor contains the following PL/SQL code:

```
create or replace function moy_sal return float
is
emp employee%rowtype;
cursor c2 is select * from employee where age > 20;
somme float := 0;
moy float;
i integer := 0;
begin
open c2;
loop
    fetch c2 into emp;
    exit when c2%notfound;
    i := i + 1;
    somme := somme + emp.salaire;
end loop;
moy := somme /i;
close c2;
return moy;
end moy_sal;
```

5) Ecrire une fonction « NB\_Moy Sal » qui renvoie le nombre des employés ayant un salaire supérieur au salaire moyen (utiliser la fonction Moy\_Sal).



6. Ecrire une procédure qui enregistre dans un tableau la liste des noms sous forme «##NomE@@» des employés participés dans le projet « Gestion des commandes » utiliser la fonction `modification_mot`:

