
Compte Rendu : TPS DEV BD
Mounji Said (F2) 2023-2024



Tout le code se trouve sur cette Github Repository :
<https://github.com/TheSkyAboveTheSky/TPs-PLSQL-Said-Mounji-F2-ISIMA>

Réalisé par :

MOUNJI SAID

Table des figures

1.1	Les tables n'existent pas avant l'initialisation	3
1.2	Initialisation des tables	3
1.3	Code de l'Insertion de l'employee MOUNJI	3
1.4	Résultat du code de l'insertion de l'employee MOUNJI	3
1.5	Code pour question 1	4
1.6	Résultat pour question 1	4
1.7	Tableau TEMP	5
1.8	Code pour question 2	5
1.9	Résultat pour question 2	5
1.10	Code pour question 3	6
1.11	Résultat pour question 3	6
1.12	Code pour question 4	7
1.13	Résultat pour question 4	7
1.14	Code pour question 5	8
1.15	Résultat pour question 5	8
2.1	Code pour question A.1	10
2.2	Résultat pour question A.1	10
2.3	le test du question A.1	10
2.4	Code pour la création de la table SalIntervalle_mounji	11
2.5	Résultat pour la création de la table SalIntervalle_mounji	11
2.6	Screen du code	12
2.7	Test 1 du Question A.2 du TP3	13
2.8	Test 2 du Question A.2 du TP3	14
2.9	Screen pour le code	15
2.10	Test 1 du Question A.3 du TP3	16
2.11	Test 2 du Question A.3 du TP3	17
2.12	Création de la table MOUNJI	18
2.13	Capture d'écran du Code	19
2.14	Liste des Tableaux Avant l'exécution	20
2.15	Liste des Tableaux Après l'exécution	20
2.16	Liste des Tableaux Après 4 exécutions	21
3.1	Création du package	24
3.2	Création du Body du package	24
3.3	Code du trigger	25
3.4	Test 1	25
3.5	Test 2	25
3.6	Code du trigger	26
3.7	Ajouter département numéro 65 pour le Test	27
3.8	Test 1	27
3.9	Test 2	27
3.10	Code pour question 3	28
3.11	Résultat pour question 3	28
3.12	Question 4	29
3.13	Question 5	30
3.14	Question 6	30
3.15	Création de la table STATS_mounji	32
3.16	Code du trigger	33
3.17	Résultat du Test du trigger	33

3.18 Test Avec FOR EACH ROW	34
3.19 Test Sans FOR EACH ROW	34
3.20 Code du trigger	36
3.21 Ajouter Un Clerk pour le Test	36
3.22 Test	36

Table des matières

1 TP Numéro 2 :	1
1.1 Création de la base de données	1
1.1.1 Question :	1
1.1.2 Code :	1
1.1.3 Resultat :	3
1.1.3.1 Initialisation des Tables :	3
1.1.3.2 Insertion :	3
1.2 Question 1 :	3
1.2.1 Question :	3
1.2.2 Code :	4
1.2.3 Resultat :	4
1.3 Question 2 :	4
1.3.1 Question :	4
1.3.2 Code :	5
1.3.3 Resultat :	5
1.4 Question 3 :	5
1.4.1 Question :	5
1.4.2 Code :	6
1.4.3 Resultat :	6
1.5 Question 4 :	6
1.5.1 Question :	6
1.5.2 Code :	7
1.5.3 Resultat :	7
1.6 Question 5 :	7
1.6.1 Question :	7
1.6.2 Code :	8
1.6.3 Resultat :	8
2 TP Numéro 3 :	9
2.1 Question A.1 :	9
2.1.1 Question :	9
2.1.2 Implémentation :	9
2.1.2.1 Code :	9
2.1.2.2 Resultat :	10
2.1.3 Test :	10
2.1.3.1 Code :	10
2.1.3.2 Resultat :	10
2.2 Question A.2 :	11
2.2.1 Question :	11
2.2.2 Création de la table SalIntervalle_mounji :	11
2.2.2.1 Code :	11
2.2.2.2 Resultat :	11
2.2.3 Implémentatation	12
2.2.3.1 Code :	12
2.2.3.2 capture d'écran :	12
2.2.4 Test	13
2.2.4.1 Test Numéro 1 :	13
2.2.4.2 Test Numéro 2 :	13

2.2.1	Code : 13	2.2.2	Réultat : 14	
2.3	Question A.3 :			14
2.3.1	Question :			14
2.3.2	Code :			15
2.3.3	Capture d'écran :			15
2.3.4	Test :			16
2.3.4.1	Test Numéro 1 :			16
2.3.4.2	Test Numéro 2 :			16
2.3.4.3	Réultat : 16			
2.3.4.4	Code : 16			
2.3.4.5	Resultat : 17			
2.4	Question B :			17
2.4.1	Question :			17
2.4.2	Création de la Table :			18
2.4.2.1	Code :			18
2.4.3	Implémentation :			19
2.4.3.1	Code :			19
2.4.3.2	Capture d'écran :			19
2.4.3.3	Liste des Tableaux Avant l'exécution :			20
2.4.3.4	Liste des Tableaux Après l'exécution :			20
2.4.3.5	Liste des Tableaux Après 4 exécutions :			21
3	TP Numéro 4 :			22
3.1	Question Package :			22
3.1.1	Question :			22
3.1.2	Code :			22
3.1.3	Resultat :			24
3.2	Question 1 :			24
3.2.1	Question :			24
3.2.2	Code :			24
3.2.3	Resultat :			25
3.2.3.1	Code :			25
3.2.3.2	Test :			25
3.3	Question 2 :			26
3.3.1	Question :			26
3.3.2	Code :			26
3.3.3	Resultat :			26
3.3.3.1	Code :			26
3.3.3.2	Test :			27
3.3.4	Appliquer département numéro 65 pour le Test : 27			
3.3.4.1	Code :			
3.3.4.2	Resultat :			
3.4	Question 3 :			28
3.4.1	Question :			28
3.4.2	Code :			28
3.4.3	Resultat :			28
3.5	Question 4 :			29
3.5.1	Question :			29
3.5.2	Code :			29
3.5.3	Resultat :			29
3.6	Question 5 :			29
3.6.1	Question :			29
3.6.2	Code :			30
3.6.3	Resultat :			30
3.7	Question 6 :			30
3.7.1	Question :			30
3.7.2	Code :			30
3.7.3	Resultat :			30
3.8	Question 7 :			31
3.8.1	Question :			31
3.8.2	Création de la Table STATS_mounji :			31
3.8.2.1	Resultat de la création de la table :			32
3.8.3	Question A :			33
3.8.3.1	Resultat :			33

3.8.4	Question B :	34
3.8.4.1	Resultat :	34
3.9	Question 8 :	34
3.9.1	Question :	34
3.9.2	Code :	35
3.9.3	Resultat :	35

Chapitre 1

TP Numéro 2 :

1.1 Création de la base de données

1.1.1 Question :

Question

Télécharger et lancer le script TP2script.sql sur l'ENT.

Insérer une ligne dans la table emp avec votre nom :

```
1 insert into emp values (7000, 'votre_nom', 'SALESMAN', 7566, to_date('
2   17-12-1980',
3 'dd-mm-yyyy'), 3500, NULL, 20);
4 Commit;
```

1.1.2 Code :

```
sql

1 drop table emp ;
2 drop Table Dept ;
3
4
5 create table dept(
6   deptno NUMBER(2) primary key,
7   dname VARCHAR2(14),
8   loc  VARCHAR2(13));
9
10 create table emp(
11   empno      number(4) primary key,
12   ename      varchar2(10),
13   job        varchar2(9),
14   mgr        number(4),
15   hiredate   date,
16   sal         number(7,2),
17   comm        number(7,2),
18   deptno     number(2) references dept) ;
19
20 drop table temp ;
21
22 create table temp(
23   num_col1   number(9,4),
24   num_col2   number(9,4),
25   char_col   char(55)) ;
```

```
sql
```

```
1 insert into dept values
2 (10 , 'ACCOUNTING' , 'NEW YORK') ;
3
4 insert into dept values
5 (20 , 'RESEARCH' , 'DALLAS') ;
6 insert into dept values
7 (30 , 'SALES' , 'CHICAGO') ;
8
9 insert into emp values
10 (7369 , 'SMITH' , 'CLERK' , 7902 , to_date('17-12-1980' , 'dd-mm-yyyy') , 800 , NULL , 20) ;
11
12 insert into emp values
13 (7499 , 'ALLEN' , 'SALESMAN' , 7698 , to_date('20-2-1981' , 'dd-mm-yyyy') , 1600 , 300 , 30) ;
14
15 insert into emp values
16 (7521 , 'WARD' , 'SALESMAN' , 7698 , to_date('22-2-1981' , 'dd-mm-yyyy') , 1250 , 500 , 30) ;
17
18 insert into emp values
19 (7566 , 'JONES' , 'MANAGER' , 7839 , to_date('2-4-1981' , 'dd-mm-yyyy') , 2975 , NULL , 20) ;
20
21 insert into emp values
22 (7654 , 'MARTIN' , 'SALESMAN' , 7698 , to_date('28-9-1981' , 'dd-mm-yyyy') , 1250 , 1400 , 30) ;
23
24
25 insert into emp values
26 (7698 , 'BLAKE' , 'MANAGER' , 7839 , to_date('1-5-1981' , 'dd-mm-yyyy') , 2850 , NULL , 30) ;
27
28 insert into emp values
29 (7782 , 'CLARK' , 'MANAGER' , 7839 , to_date('9-6-1981' , 'dd-mm-yyyy') , 2450 , NULL , 10) ;
30
31 insert into emp values
32 (7788 , 'SCOTT' , 'ANALYST' , 7566 , to_date('13-7-1987' , 'dd-mm-yyyy') - 85 , 3000 , NULL , 20) ;
33
34 insert into emp values
35 (7839 , 'KING' , 'PRESIDENT' , NULL , to_date('17-11-1981' , 'dd-mm-yyyy') , 5000 , NULL , 10) ;
36
37 insert into emp values
38 (7844 , 'TURNER' , 'SALESMAN' , 7698 , to_date('8-9-1981' , 'dd-mm-yyyy') , 1500 , 0 , 30) ;
39
40 insert into emp values
41 (7876 , 'ADAMS' , 'CLERK' , 7788 , to_date('13-7-1987' , 'dd-mm-yyyy') - 51 , 1100 , NULL , 20) ;
42
43 insert into emp values
44 (7900 , 'JAMES' , 'CLERK' , 7698 , to_date('3-12-1981' , 'dd-mm-yyyy') , 950 , NULL , 30) ;
45
46 insert into emp values
47 (7902 , 'FORD' , 'ANALYST' , 7566 , to_date('3-12-1981' , 'dd-mm-yyyy') , 3000 , NULL , 20) ;
48
49 insert into emp values
50 (7934 , 'MILLER' , 'CLERK' , 7782 , to_date('23-1-1982' , 'dd-mm-yyyy') , 1300 , NULL , 10) ;
51
52 commit;
```

```
sql
```

```
1 insert into emp values (7000 , 'MOUNJI' , 'SALESMAN' , 7566 , to_date('17-08-2001' , 'dd-mm-yyyy')
2 , 3500 , NULL , 20) ;
2 Commit;
```

1.1.3 Resultat :

1.1.3.1 Initialisation des Tables :

```
SQL> select * from emp;
select * from emp
      *
ERREUR a la ligne 1 :
ORA-00942: Table ou vue inexistante
```

FIGURE 1.1 – Les tables n'existent pas avant l'initialisation

```
SQL> commit;
Validation effectuée.

SQL> select * from emp;
-----+-----+-----+-----+-----+-----+
EMPNO|ENAME |JOB   |MGR   |HIREDATE|SAL    |COMM   |
-----+-----+-----+-----+-----+-----+
DEPTNO
-----+-----+
7369|SMITH |CLERK|7902 |17/12/80|800    |
      |20    |
7499|ALLEN |SALESMAN|7698 |20/02/81|1600   |300    |
      |30    |
7521|WARD  |SALESMAN|7698 |22/02/81|1250   |500    |
      |30    |
-----+-----+-----+-----+-----+-----+
EMPNO|ENAME |JOB   |MGR   |HIREDATE|SAL    |COMM   |
-----+-----+-----+-----+-----+-----+
DEPTNO
-----+-----+
7566|JONES |MANAGER|7839 |02/04/81|2975   |
      |20    |
7654|MARTIN|SALESMAN|7698 |28/09/81|1250   |1400   |
      |
-----+-----+-----+-----+-----+-----+
```

FIGURE 1.2 – Initialisation des tables

1.1.3.2 Insertion :

```
insert into emp values (7000,'MOUNJI','SALESMAN',7566,to_date('17-08-2001','dd-mm-yyyy'),3500,NULL,20);
1 ligne creee.

SQL> Commit;
Validation effectuée.

SQL> ■
```

FIGURE 1.3 – Code de l'Insertion de l'employee MOUNJI

```
-----+-----+-----+-----+-----+
EMPNO|ENAME |JOB   |MGR   |HIREDATE|SAL    |COMM   |
-----+-----+-----+-----+-----+-----+
DEPTNO
-----+-----+
7902|FORD  |ANALYST|7566 |03/12/81|3000   |
      |20    |
7934|MILLER|CLERK |7782 |23/01/82|1300   |
      |10    |
7000|MOUNJI|SALESMAN|7566 |17/08/01|3500   |
      |20    |
-----+-----+-----+-----+-----+-----+
```

FIGURE 1.4 – Resultat du code de l'insertion de l'employee MOUNJI

1.2 Question 1 :

1.2.1 Question :

Question

Afficher à l'écran le nom , salaire, la commission de l'employé ‘MILLER’ ainsi que le nom du département dans lequel il travail.

1.2.2 Code :

```
sql

1 SET SERVEROUTPUT ON;
2 DECLARE
3     v_ename    emp.ename%TYPE;
4     v_sal      emp.sal%TYPE;
5     v_comm     emp.comm%TYPE;
6     v_dname    dept.dname%TYPE;
7 BEGIN
8     SELECT e.ename, e.sal, e.comm, d.dname
9        INTO v_ename, v_sal, v_comm, v_dname
10       FROM emp e
11      JOIN dept d ON e.deptno = d.deptno
12     WHERE e.ename = 'MILLER';
13
14     DBMS_OUTPUT.PUT_LINE('- Nom de l''employe : ' || v_ename);
15     DBMS_OUTPUT.PUT_LINE('- Salaire de l''employe : ' || v_sal);
16
17     IF v_comm IS NOT NULL THEN
18         DBMS_OUTPUT.PUT_LINE('- Commission de l''employe : ' || v_comm);
19     ELSE
20         DBMS_OUTPUT.PUT_LINE('- Pas de commission pour cet employe.');
21     END IF;
22
23     DBMS_OUTPUT.PUT_LINE('- Nom du departement : ' || v_dname);
24 END;
25 /
```

1.2.3 Resultat :

```
SET SERVEROUTPUT ON;
SQL> DECLARE
2     v_ename    emp.ename%TYPE;
3     v_sal      emp.sal%TYPE;
4     v_comm     emp.comm%TYPE;
5     v_dname    dept.dname%TYPE;
6 BEGIN
7     SELECT e.ename, e.sal, e.comm, d.dname
8        INTO v_ename, v_sal, v_comm, v_dname
9       FROM emp e
10      JOIN dept d ON e.deptno = d.deptno
11     WHERE e.ename = 'MILLER';
12
13     DBMS_OUTPUT.PUT_LINE('- Nom de l''employe : ' || v_ename);
14     DBMS_OUTPUT.PUT_LINE('- Salaire de l''employe : ' || v_sal);
15
16     IF v_comm IS NOT NULL THEN
17         DBMS_OUTPUT.PUT_LINE('- Commission de l''employe : ' || v_comm);
18     ELSE
19         DBMS_OUTPUT.PUT_LINE('- Pas de commission pour cet employe.');
20     END IF;
21
22     DBMS_OUTPUT.PUT_LINE('- Nom du departement : ' || v_dname);
23 END;
24 /
```

- Nom de l'employe : MILLER
- Salaire de l'employe : 1300
- Pas de commission pour cet employe.
- Nom du departement : ACCOUNTING

Procedure PL/SQL terminee avec succes.

SQL> SQL> █

FIGURE 1.6 – Resultat pour question 1

FIGURE 1.5 – Code pour question 1

1.3 Question 2 :

1.3.1 Question :

Question

Ecrire un programme PL/SQL permettant d'insérer dans la table Temp les 100 lignes suivantes :

Temp		
Num_COL1	Num_COL2	Char_COL
1	100	1 est impair
2	200	2 est paire
3	300	3 est impair
...
...
100	10000	100 est paire

FIGURE 1.7 – Tableau TEMP

1.3.2 Code :

```
sql

1 DECLARE
2   v_num_col1 Temp.num_col1%TYPE;
3   v_num_col2 Temp.num_col2%TYPE;
4   v_char_col Temp.char_col%TYPE;
5 BEGIN
6   FOR i IN 1..100 LOOP
7     v_num_col1 := i;
8     v_num_col2 := i * 100;
9     v_char_col := CASE WHEN MOD(i, 2) = 0 THEN i || 'est paire' ELSE i || 'est'
10                      'impair' END;
11     INSERT INTO Temp (num_col1, num_col2, char_col)
12       VALUES (v_num_col1, v_num_col2, v_char_col);
13   END LOOP;
14   COMMIT;
15 
```

1.3.3 Résultat :

```
DECLARE
2   v_num_col1 Temp.num_col1%TYPE;
3   v_num_col2 Temp.num_col2%TYPE;
4   v_char_col Temp.char_col%TYPE;
5 BEGIN
6   FOR i IN 1..100 LOOP
7     v_num_col1 := i;
8     v_num_col2 := i * 100;
9     v_char_col := CASE WHEN MOD(i, 2) = 0 THEN i || 'est paire' ELSE i || 'est impair' END;
10    INSERT INTO Temp (num_col1, num_col2, char_col)
11      VALUES (v_num_col1, v_num_col2, v_char_col);
12  END LOOP;
13  COMMIT;
14 
```

Procedure PL/SQL terminée avec succès.

FIGURE 1.8 – Code pour question 2

The screenshot shows the Oracle SQL Developer interface with a window titled 'Samson@oracle-srv19c'. The results pane displays the output of the query. It consists of two main sections of data:

- Section 1 (Top):** Contains 100 rows labeled 1 to 100. Each row has three columns: NUM_COL1, NUM_COL2, and CHAR_COL. The CHAR_COL values are:
 - Rows 1-50: 'i est paire' (e.g., 1 est paire, 2 est paire, ..., 50 est paire)
 - Rows 51-100: 'i est impair' (e.g., 51 est impair, 52 est impair, ..., 100 est impair)
- Section 2 (Bottom):** Contains a single row labeled '100'. It has three columns: NUM_COL1, NUM_COL2, and CHAR_COL. The CHAR_COL value is '100 est paire'.

Below the results, it says '100 lignes sélectionnées.'

FIGURE 1.9 – Résultat pour question 2

1.4 Question 3 :

1.4.1 Question :

Question

Insérer dans la table Temp (sal, empno, ename) les 5 employés les mieux payés.

1.4.2 Code :

```

sql

1  DECLARE
2      CURSOR top_sal_emp_cur IS
3          SELECT sal, empno, ename
4          FROM (SELECT sal, empno, ename, RANK() OVER (ORDER BY sal DESC) AS ranking
5                  FROM Emp)
6          WHERE ranking <= 5;
7
8      v_sal emp.sal%TYPE;
9      v_empno emp.empno%TYPE;
10     v_ename emp.ename%TYPE;
11
12    BEGIN
13        FOR employee_record IN top_sal_emp_cur LOOP
14            v_sal := employee_record.sal;
15            v_empno := employee_record.empno;
16            v_ename := employee_record.ename;
17
18            INSERT INTO Temp (num_col1, num_col2, char_col)
19                VALUES (v_sal, v_empno, v_ename);
20        END LOOP;
21        COMMIT;
22    END;
23

```

1.4.3 Resultat :

```

DECLARE
2      CURSOR top_sal_emp_cur IS
3          SELECT sal, empno, ename
4          FROM (SELECT sal, empno, ename, RANK() OVER (ORDER BY sal DESC) AS ranking
5                  FROM Emp)
6          WHERE ranking <= 5;
7
8      v_sal emp.sal%TYPE;
9      v_empno emp.empno%TYPE;
10     v_ename emp.ename%TYPE;
11
12    BEGIN
13        FOR employee_record IN top_sal_emp_cur LOOP
14            v_sal := employee_record.sal;
15            v_empno := employee_record.empno;
16            v_ename := employee_record.ename;
17
18            INSERT INTO Temp (num_col1, num_col2, char_col)
19                VALUES (v_sal, v_empno, v_ename);
20        END LOOP;
21        COMMIT;
22    END;
23
Procedure PL/SQL terminee avec succes.

```

FIGURE 1.10 – Code pour question 3

```

SQL> select * from Temp;
-----+-----+-----+
NUM_COL1 | NUM_COL2 | CHAR_COL |
-----+-----+-----+
      5000 |    7839 | KING
      3500 |    7000 | MOUNJI
      3000 |    7788 | SCOTT
      3000 |    7902 | FORD
      2975 |    7566 | JONES
-----+-----+-----+
      1   |    100  | 1 est impair
      2   |    200  | 2 est paire
      3   |    300  | 3 est impair
      4   |    400  | 4 est paire
      5   |    500  | 5 est impair
      6   |    600  | 6 est paire
-----+-----+-----+
      7   |    700  | 7 est impair
      8   |    800  | 8 est paire
      9   |    900  | 9 est impair
     10   |   1000 | 10 est paire
     11   |   1100 | 11 est impair
     12   |   1200 | 12 est paire
     13   |   1300 | 13 est impair
     14   |   1400 | 14 est paire
-----+-----+-----+

```

FIGURE 1.11 – Resultat pour question 3

1.5 Question 4 :

1.5.1 Question :

Question

Récupérer dans une table Temp tous les employés dont les revenus mensuels (salaire + commission) sont supérieurs à \$2000.

1.5.2 Code :

```

sql

1  DECLARE
2      CURSOR sal_plus_comm_emp_cursor IS
3          SELECT NVL(e.sal,0) + NVL(e.comm,0) as sal_plus_commission , e.empno , e.ename
4          FROM Emp e
5          WHERE NVL(e.sal, 0) + NVL(e.comm, 0) > 2000;
6
7      v_sal_plus_comm emp.sal%TYPE;
8      v_empno emp.empno%TYPE;
9      v_ename emp.ename%TYPE;
10
11     BEGIN
12         FOR employee_record IN sal_plus_comm_emp_cursor LOOP
13             v_sal_plus_comm := employee_record.sal_plus_commission;
14             v_empno := employee_record.empno;
15             v_ename := employee_record.ename;
16
17             INSERT INTO Temp (num_col1, num_col2, char_col)
18             VALUES (v_sal_plus_comm, v_empno, v_ename);
19         END LOOP;
20         COMMIT;
21     END;
22 /

```

1.5.3 Resultat :

```

DECLARE
2   CURSOR sal_plus_comm_emp_cursor IS
3       SELECT NVL(e.sal,0) + NVL(e.comm,0) as sal_plus_commission , e.empno , e.ename
4       FROM Emp e
5       WHERE NVL(e.sal, 0) + NVL(e.comm, 0) > 2000;
6
7   v_sal_plus_comm emp.sal%TYPE;
8   v_empno emp.empno%TYPE;
9   v_ename emp.ename%TYPE;
10
11  BEGIN
12    FOR employee_record IN sal_plus_comm_emp_cursor LOOP
13      v_sal_plus_comm := employee_record.sal_plus_commission;
14      v_empno := employee_record.empno;
15      v_ename := employee_record.ename;
16
17      INSERT INTO Temp (num_col1, num_col2, char_col)
18      VALUES (v_sal_plus_comm, v_empno, v_ename);
19    END LOOP;
20    COMMIT;
21  END;
22 /
Procedure PL/SQL terminee avec succes.
SQL> 
```

NUM_COL1	NUM_COL2	CHAR_COL
95	9500	95 est impair
96	9600	96 est paire
97	9700	97 est impair
98	9800	98 est paire
99	9900	99 est impair
100	10000	100 est paire
2975	7566	JONES
2650	7654	MARTIN
2850	7698	BLAKE
2450	7782	CLARK
3000	7788	SCOTT
5000	7839	KING
3000	7902	FORD
3500	7000	MOUNJI

113 lignes selectionnees.

FIGURE 1.12 – Code pour question 4

FIGURE 1.13 – Resultat pour question 4

1.6 Question 5 :

1.6.1 Question :

Question

Insérer dans une table Temp(sal, ename) le premier employé qui a un salaire supérieur à \$4000 et qui est plus haut dans la chaîne de la commande que l'employé 7902.

1.6.2 Code :

```

sql

1 SET SERVEROUTPUT ON;
2 DECLARE
3     empno_base    emp.empno%TYPE;
4     emp_record   emp%ROWTYPE;
5     v_sal        emp.sal%TYPE;
6     v_ename      emp.ename%TYPE;
7 BEGIN
8     empno_base := 7902;
9
10    WHILE TRUE LOOP
11        BEGIN
12            SELECT * INTO emp_record
13            FROM emp
14            WHERE empno = (SELECT mgr FROM emp WHERE empno = empno_base);
15
16            IF emp_record.sal > 4000 THEN
17                v_sal := emp_record.sal;
18                v_ename := emp_record.ename;
19                INSERT INTO Temp (num_col1, char_col) VALUES (v_sal, v_ename);
20            END IF;
21
22            empno_base := emp_record.empno;
23        EXCEPTION
24            WHEN NO_DATA_FOUND THEN
25                EXIT;
26            END;
27        END LOOP;
28    END;
29/

```

1.6.3 Resultat :

```

SQL> DECLARE
2     empno_base emp.empno%TYPE;
3     emp_record emp%ROWTYPE;
4     v_sal      emp.sal%TYPE;
5     v_ename    emp.ename%TYPE;
6     BEGIN
7         empno_base := 7902;
8
9         WHILE TRUE LOOP
10            BEGIN
11                SELECT * INTO emp_record
12                FROM emp
13                WHERE empno = (SELECT mgr FROM emp WHERE empno = empno_base);
14
15                IF emp_record.sal > 4000 THEN
16                    v_sal := emp_record.sal;
17                    v_ename := emp_record.ename;
18                    INSERT INTO Temp (num_col1, char_col) VALUES (v_sal, v_ename);
19                END IF;
20
21                empno_base := emp_record.empno;
22            EXCEPTION
23                WHEN NO_DATA_FOUND THEN
24                    EXIT;
25            END;
26        END LOOP;
27    END;
28/

```

NUM_COL1	NUM_COL2 CHAR_COL
95	9500 95 est impair
96	9600 96 est paire
97	9700 97 est impair
98	9800 98 est paire
99	9900 99 est impair
100	10000 100 est paire
5000	KING
2975	7566 JONES
2650	7654 MARTIN
2850	7698 BLAKE
2450	7782 CLARK

NUM_COL1	NUM_COL2 CHAR_COL
3000	7788 SCOTT
5000	7839 KING
3000	7902 FORD
3500	7000 MOUNJI

114 lignes selectionnees.

SQL>

FIGURE 1.15 – Resultat pour question 5

FIGURE 1.14 – Code pour question 5

Chapitre 2

TP Numéro 3 :

2.1 Question A.1 :

2.1.1 Question :

Question

Procédure `createdept_votrenom(numéro_dept, dept_name, localisation)` : permettant de créer de nouveaux départements dans la table Dept. Vérifier si le numéro_dept existe déjà dans la table.

2.1.2 Implémentation :

2.1.2.1 Code :

sql

```
1 SET SERVEROUTPUT ON;
2 CREATE OR REPLACE PROCEDURE createdept_mounji (
3     p_numero_dept IN dept.deptno%TYPE,
4     p_dept_name    IN dept.dname%TYPE,
5     p_localisation IN dept.loc%TYPE
6 )
7 IS
8     v_count NUMBER;
9 BEGIN
10     SELECT COUNT(*) INTO v_count FROM Dept WHERE deptno = p_numero_dept;
11     IF v_count = 0 THEN
12         INSERT INTO Dept (deptno, dname, loc)
13             VALUES (p_numero_dept, p_dept_name, p_localisation);
14         DBMS_OUTPUT.PUT_LINE('Département cree avec succes.');
15     ELSE
16         DBMS_OUTPUT.PUT_LINE('Un departement avec le numero ' || p_numero_dept || ' existe deja.');
17     END IF;
18 END ;
19 /
```

```

SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE createdept_mounji (
SQL> 2      p_numero_dept IN dept.deptno%TYPE,
3      p_dept_name   IN dept.dname%TYPE,
4      p_localisation IN dept.loc%TYPE
5  )
6  IS
7      v_count NUMBER;
8  BEGIN
9      SELECT COUNT(*) INTO v_count FROM Dept WHERE deptno = p_numero_dept;
10     IF v_count > 0 THEN
11         INSERT INTO Dept (deptno, dname, loc)
12             VALUES (p_numero_dept, p_dept_name, p_localisation);
13         DBMS_OUTPUT.PUT_LINE('Departement cree avec succes.');
14     ELSE
15         DBMS_OUTPUT.PUT_LINE('Un departement avec le numero ' || p_numero_dept || ' existe deja.');
16     END IF;
17 END;
18 /

```

Procedure creee.

FIGURE 2.1 – Code pour question A.1

SQL> select * from dept;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
50	IT	New York

SQL> ■

FIGURE 2.2 – Resultat pour question A.1

2.1.2.2 Résultat :

2.1.3 Test :

2.1.3.1 Code :

sql

```

1 BEGIN
2     createdept_mounji(50, 'IT', 'New York');
3 END;
4 /

```

2.1.3.2 Résultat :

```

BEGIN
2     createdept_mounji(50, 'IT', 'New York');
3 END;
4 /
Departement cree avec succes.

Procedure PL/SQL terminee avec succes.

```

SQL> ■

FIGURE 2.3 – le test du question A.1

2.2 Question A.2 :

2.2.1 Question :

Question

Fonction `salok_votrenom(job, salaire)` Return Number : fonction qui vérifie que le salaire d'un job est dans un intervalle donné (nécessité d'une table SalIntervalle (job, lsal, hsal)). Retourner 1 si c'est le cas, sinon 0.

Mise en place : exécuter le script SQL suivant pour créer et remplir la table SalIntervalle_votrenom.

```
1 create table SalIntervalle_votrenom (job varchar2(9), lsal number(7,2), hsal
2   number(7,2));
3
4 insert into SalIntervalle_votrenom values ('ANALYST', 2500, 3000);
5 insert into SalIntervalle_votrenom values ('CLERK', 900, 1300);
6 insert into SalIntervalle_votrenom values ('MANAGER', 2400, 3000);
7 insert into SalIntervalle_votrenom values ('PRESIDENT', 4500, 4900);
8 insert into SalIntervalle_votrenom values ('SALESMAN', 1200, 1700);
```

2.2.2 Crédation de la table SalIntervalle_mounji :

2.2.2.1 Code :

sql

```
1 create table SalIntervalle_mounji (job varchar2(9), lsal number(7,2), hsal number(7,2));
2 insert into SalIntervalle_mounji values ('ANALYST', 2500, 3000) ;
3 insert into SalIntervalle_mounji values ('CLERK', 900, 1300) ;
4 insert into SalIntervalle_mounji values ('MANAGER', 2400, 3000) ;
5 insert into SalIntervalle_mounji values ('PRESIDENT', 4500, 4900) ;
6 insert into SalIntervalle_mounji values ('SALESMAN', 1200, 1700) ;
```

2.2.2.2 Resultat :

```
SQL>
SQL> select * from salIntervalle_mounji;

          JOB        LSAL        HSAL
-----  -----  -----
ANALYST      2500      3000
CLERK        900       1300
MANAGER     2400      3000
PRESIDENT    4500      4900
SALESMAN    1200      1700
```

FIGURE 2.4 – Code pour la création de la table SalIntervalle_mounji

FIGURE 2.5 – Résultat pour la création de la table SalIntervalle_mounji

2.2.3 Implémentation

2.2.3.1 Code :

```
sql

1 CREATE OR REPLACE FUNCTION salok_mounji (
2     p_job      IN VARCHAR2 ,
3     p_salaire  IN NUMBER
4 )
5 RETURN NUMBER
6 IS
7     v_count NUMBER;
8 BEGIN
9     SELECT COUNT(*)
10    INTO v_count
11   FROM SalIntervalle_mounji im
12  WHERE im.job = p_job
13  AND p_salaire BETWEEN im.lsal AND im.hsal;
14
15     RETURN CASE WHEN v_count > 0 THEN 1 ELSE 0 END;
16 END;
17 /
```

2.2.3.2 Aapture d'écran :

Procedure PL/SQL terminée avec succès.

```
CREATE OR REPLACE FUNCTION salok_mounji (
2     p_job      IN VARCHAR2 ,
3     p_salaire  IN NUMBER
4 )
5 RETURN NUMBER
6 IS
7     v_count NUMBER;
8 BEGIN
9     SELECT COUNT(*)
10    INTO v_count
11   FROM SalIntervalle_mounji im
12  WHERE im.job = p_job
13  AND p_salaire BETWEEN im.lsal AND im.hsal;
14
15     RETURN CASE WHEN v_count > 0 THEN 1 ELSE 0 END;
16 END;
17 /
```

Fonction créée.

FIGURE 2.6 – Screen du code

2.2.4 Test

2.2.4.1 Test Numéro 1 :

2.2.4.1.1 Code :

```
sql

1 SET SERVEROUTPUT ON;
2 DECLARE
3     result NUMBER;
4 BEGIN
5     result := salok_mounji('CLERK', 5000);
6     DBMS_OUTPUT.PUT_LINE('Resultat:' || result);
7 END;
8 /
```

2.2.4.1.2 Resultat :

```
SET SERVEROUTPUT ON;
SQL> DECLARE
 2     result NUMBER;
 3 BEGIN
 4     result := salok_mounji('CLERK', 5000);
 5     DBMS_OUTPUT.PUT_LINE('Resultat: ' || result);
 6 END;
 7 /
Resultat: 0
```

Procedure PL/SQL terminee avec succes.

FIGURE 2.7 – Test 1 du Question A.2 du TP3

2.2.4.2 Test Numéro 2 :

2.2.4.2.1 Code :

```
sql

1 SET SERVEROUTPUT ON;
2 DECLARE
3     result NUMBER;
4 BEGIN
5     result := salok_mounji('CLERK', 900);
6     DBMS_OUTPUT.PUT_LINE('Resultat:' || result);
7 END;
8 /
```

```
SET SERVEROUTPUT ON;
SQL> DECLARE
  2      result NUMBER;
  3  BEGIN
  4      result := salok_mounji('CLERK', 900);
  5      DBMS_OUTPUT.PUT_LINE('Resultat: ' || result);
  6  END;
  7 /
Resultat: 1
```

Procedure PL/SQL terminee avec succes.

SQL> █

FIGURE 2.8 – Test 2 du Question A.2 du TP3

2.2.4.2.2 Resultat :

2.3 Question A.3 :

2.3.1 Question :

Question

Procédure raisesalary_votrenom (emp_id, amount) : permettant d'augmenter le salaire d'un employé d'un montant donné si le salaire augmenté est dans l'intervalle concerné (la fonction salok_votrenom). Sinon, afficher un message d'erreur.

2.3.2 Code :

```
sql

1 CREATE OR REPLACE PROCEDURE raisesalary_mounji (
2     p_emp_id    IN NUMBER,
3     p_amount    IN NUMBER
4 )
5 IS
6     v_sal_courant  emp.sal%TYPE;
7     v_sal_nv       emp.sal%TYPE;
8     v_emp_job      emp.job%TYPE;
9
10    BEGIN
11        SELECT sal, job INTO v_sal_courant, v_emp_job
12        FROM emp
13        WHERE empno = p_emp_id;
14
15        v_sal_nv := v_sal_courant + p_amount;
16
17        IF salok_mounji(v_emp_job, v_sal_nv) = 1 THEN
18            UPDATE emp
19            SET sal = v_sal_nv, job = v_emp_job
20            WHERE empno = p_emp_id;
21
22            DBMS_OUTPUT.PUT_LINE('Salaire augmenté avec succès.');
23            commit;
24        ELSE
25            DBMS_OUTPUT.PUT_LINE('Erreur : Le nouveau salaire est en dehors de l''intervalle autorisé.');
26        END IF;
27    END;
28
```

2.3.3 capture d'écran :

```
CREATE OR REPLACE PROCEDURE raisesalary_mounji (
2     p_emp_id    IN NUMBER,
3     p_amount    IN NUMBER
4 )
5 IS
6     v_sal_courant  emp.sal%TYPE;
7     v_sal_nv       emp.sal%TYPE;
8     v_emp_job      emp.job%TYPE;
9
10    BEGIN
11        SELECT sal, job INTO v_sal_courant, v_emp_job
12        FROM emp
13        WHERE empno = p_emp_id;
14
15        v_sal_nv := v_sal_courant + p_amount;
16
17        IF salok_mounji(v_emp_job, v_sal_nv) = 1 THEN
18            UPDATE emp
19            SET sal = v_sal_nv, job = v_emp_job
20            WHERE empno = p_emp_id;
21
22            DBMS_OUTPUT.PUT_LINE('Salaire augmenté avec succès.');
23            commit;
24        ELSE
25            DBMS_OUTPUT.PUT_LINE('Erreur : Le nouveau salaire est en dehors de l''intervalle autorisé.');
26        END IF;
27    END;
28
```

FIGURE 2.9 – Screen pour le code

2.3.4 Test

2.3.4.1 Test Numéro 1 :

2.3.4.1.1 Code :

```
sql  
1 SET SERVEROUTPUT ON;  
2 BEGIN  
3     raisesalary_mounji(7369, 4000);  
4 END;  
5 /
```

2.3.4.1.2 Resultat :

```
SET SERVEROUTPUT ON;  
SQL> BEGIN  
2     raisesalary_mounji(7369, 4000);  
3 END;  
4 /  
Erreur : Le nouveau salaire est en dehors de l'intervalle autorise.  
  
Procedure PL/SQL terminee avec succes.
```

FIGURE 2.10 – Test 1 du Question A.3 du TP3

2.3.4.2 Test Numéro 2 :

2.3.4.2.1 Code :

```
sql  
1 SET SERVEROUTPUT ON;  
2 BEGIN  
3     raisesalary_mounji(7369, 100);  
4 END;  
5 /
```

```

SET SERVEROUTPUT ON;
SQL> BEGIN
 2      raisesalary_mounji(7369, 100);
 3  END;
 4 /
Salaire augmente avec succes.

```

Procedure PL/SQL terminée avec succès.

FIGURE 2.11 – Test 2 du Question A.3 du TP3

2.3.4.2.2 Résultat :

2.4 Question B :

2.4.1 Question :

Question

Implémenter un bloc PL/SQL qui fait un backup de toutes les tables de l'utilisateur connecté en les copiant. La copie d'une table aura le nom `nomtable_old`, ex. la copie de la table `emp` -> `emp_old`. On peut copier la table `emp` dans la table `emp_old` en la créant par la commande,

```

1 CREATE TABLE emp_old AS
2 SELECT * FROM emp;

```

La table `user_tables` donne toutes les tables (attribut `table_name`) de l'utilisateur connecté. Si la copie d'une table existe déjà, remplacer l'ancienne copie par la nouvelle. Nous supposons qu'une table ne peut pas avoir un nom qui se termine par '`_old`' en temps normal, sauf si c'est une table générée par ce programme. Par exemple, on ne peut pas avoir une table dont le nom est `emp_old_old`. Note) faire valider les commandes SQL (ex. `CREATE`, `INSERT`, ...) par commit à chaque fois si vous jugez nécessaire. Pour tester votre programme, créer la table

```

1 CREATE TABLE votrenom (id NUMBER(5));
2 INSERT INTO votrenom VALUES (1);
3 INSERT INTO votrenom VALUES (2);
4 INSERT INTO votrenom VALUES (3);

```

*** Exécuter plusieurs fois votre programme pour vérifier les tables backup se renouvellent (par exemple il ne faut pas avoir la table `votrenom_old_old`).

2.4.2 Cration de la Table :

2.4.2.1 Code :

```
sql
```

```
1 CREATE TABLE mounji (id NUMBER(5));
2 INSERT INTO mounji VALUES (1);
3 INSERT INTO mounji VALUES (2);
4 INSERT INTO mounji VALUES (3);
```

```
CREATE TABLE mounji (id NUMBER(5));
INSERT INTO mounji VALUES (1);
INSERT INTO mounji VALUES (2);
```

Table creeee.

```
SQL>
1 ligne creeee.
```

```
SQL>
1 ligne creeee.
```

```
SQL> INSERT INTO mounji VALUES (3);
```

1 ligne creeee.

```
SQL> select * from mounji;
```

ID
1
2
3

FIGURE 2.12 – Cration de la table MOUNJI

2.4.3 Implémentation

2.4.3.1 Code :

```
sql

1 SET SERVEROUTPUT ON;
2 DECLARE
3     v_table_name VARCHAR2(30);
4     v_backup_table_name VARCHAR2(30);
5     v_table_exists NUMBER;
6 BEGIN
7     FOR table_rec IN (SELECT table_name FROM user_tables WHERE table_name LIKE '%_OLD')
8         LOOP
9             v_table_name := table_rec.table_name;
10            SELECT COUNT(*) INTO v_table_exists FROM user_tables WHERE table_name = v_table_name
11            ;
12            IF v_table_exists > 0 THEN
13                EXECUTE IMMEDIATE 'DROP TABLE' || v_table_name;
14                DBMS_OUTPUT.PUT_LINE('Backup Table' || v_table_name || ' supprimée.');
15            END IF;
16            COMMIT;
17        END LOOP;
18        FOR table_rec IN (SELECT table_name from user_tables) LOOP
19            v_table_name := table_rec.table_name;
20            v_backup_table_name := v_table_name || '_OLD';
21            EXECUTE IMMEDIATE 'CREATE TABLE' || v_backup_table_name || ' AS SELECT * FROM ' || v_table_name;
22            DBMS_OUTPUT.PUT_LINE('Backup Table' || v_backup_table_name || ' créee.');
23        END LOOP;
24        COMMIT;
25    END;
26 /

```

2.4.3.2 Capture d'écran :

```
SET SERVEROUTPUT ON;
SQL> DECLARE
 2     v_table_name VARCHAR2(30);
 3     v_backup_table_name VARCHAR2(30);
 4     v_table_exists NUMBER;
BEGIN
 5     FOR table_rec IN (SELECT table_name FROM user_tables WHERE table_name LIKE '%_OLD') LOOP
 6         v_table_name := table_rec.table_name;
 7         SELECT COUNT(*) INTO v_table_exists FROM user_tables WHERE table_name = v_table_name;
 8         IF v_table_exists > 0 THEN
 9             EXECUTE IMMEDIATE 'DROP TABLE ' || v_table_name;
10             DBMS_OUTPUT.PUT_LINE('Backup Table ' || v_table_name || ' supprimée.');
11         END IF;
12         COMMIT;
13     END LOOP;
14     FOR table_rec IN (SELECT table_name from user_tables) LOOP
15         v_table_name := table_rec.table_name;
16         v_backup_table_name := v_table_name || '_OLD';
17         EXECUTE IMMEDIATE 'CREATE TABLE ' || v_backup_table_name || ' AS SELECT * FROM ' || v_table_name;
18         DBMS_OUTPUT.PUT_LINE('Backup Table ' || v_backup_table_name || ' créee.');
19     END LOOP;
20     COMMIT;
21 END;
22 /

```

FIGURE 2.13 – Capture d'écran du Code

2.4.3.3 Liste des Tableaux Avant l'exécution :

```
SQL> select table_name from user_tables;  
  
TABLE_NAME  
-----  
MOUNJI  
DEPT  
EMP  
TEMP  
SALINTERVALLE_MOUNJI  
  
SQL> █
```

FIGURE 2.14 – Liste des Tableaux Avant l'exécution

2.4.3.4 Liste des Tableaux Après l'exécution :

```
Backup Table DEPT_OLD supprimee.  
Backup Table EMP_OLD supprimee.  
Backup Table MOUNJI_OLD supprimee.  
Backup Table SALINTERVALLE_MOUNJI_OLD supprimee.  
Backup Table TEMP_OLD supprimee.  
Backup Table MOUNJI_OLD creeee.  
Backup Table DEPT_OLD creeee.  
Backup Table EMP_OLD creeee.  
Backup Table TEMP_OLD creeee.  
Backup Table SALINTERVALLE_MOUNJI_OLD creeee.  
  
Procedure PL/SQL terminee avec succes.  
  
SQL> █
```

FIGURE 2.15 – Liste des Tableaux Après l'exécution

2.4.3.5 Liste des Tableaux Après 4 exécutions :

```
SQL> select table_name from user_tables;

TABLE_NAME
-----
MOUNJI
DEPT_OLD
MOUNJI_OLD
EMP_OLD
TEMP_OLD
SALINTERVALLE_MOUNJI_OLD
DEPT
EMP
TEMP
SALINTERVALLE_MOUNJI

10 lignes selectionnees.
```

```
SQL> █
```

FIGURE 2.16 – Liste des Tableaux Après 4 exécutions

Chapitre 3

TP Numéro 4 :

3.1 Question Package :

3.1.1 Question :

Question

Créer un package qui implémente l'interface suivante (interface = la partie spécification) :

- Curseurs : `emp_par_dep_votrenom (deptno)`. % curseur du package pour le numéro et le nom des employés dans le département donné
- Procédures : `raise_salary_votrenom`. % réutiliser la procédure `raise_salary_votrenom` du TP précédent
- `afficher_emp_votrenom (deptno)`. % afficher le numéro et le nom des employées dans le département donné, utiliser le curseur `emp_par_dep_votrenom`

3.1.2 Code :

```
sql

1  SET SERVEROUTPUT ON;
2  CREATE OR REPLACE PACKAGE mounji_package IS
3      CURSOR emp_par_dep_mounji (p_deptno NUMBER) IS
4          SELECT empno, ename FROM emp WHERE deptno = p_deptno;
5
6      PROCEDURE raisesalary_mounji (
7          p_emp_id    IN NUMBER,
8          p_amount    IN NUMBER
9      );
10
11     PROCEDURE afficher_emp_mounji(p_deptno NUMBER);
12 END mounji_package;
13 /
```

sql

```
1 SET SERVEROUTPUT ON;
2 CREATE OR REPLACE PACKAGE BODY mounji_package IS
3     PROCEDURE raisesalary_mounji (
4         p_emp_id    IN NUMBER,
5         p_amount    IN NUMBER
6     )
7     IS
8         v_sal_courant emp.sal%TYPE;
9         v_sal_nv       emp.sal%TYPE;
10        v_emp_job      emp.job%TYPE;
11    BEGIN
12        SELECT sal, job INTO v_sal_courant, v_emp_job
13        FROM emp
14        WHERE empno = p_emp_id;
15        v_sal_nv := v_sal_courant + p_amount;
16
17        IF salok_mounji(v_emp_job, v_sal_nv) = 1 THEN
18            UPDATE emp
19            SET sal = v_sal_nv, job = v_emp_job
20            WHERE empno = p_emp_id;
21
22            DBMS_OUTPUT.PUT_LINE('Salaire augmenté avec succès.');
23            commit;
24        ELSE
25            DBMS_OUTPUT.PUT_LINE('Erreur : Le nouveau salaire est en dehors de l''intervalle autorisé.');
26        END IF;
27    END raisesalary_mounji;
28
29    PROCEDURE afficher_emp_mounji(p_deptno NUMBER) IS
30        emp_rec emp_par_dep_mounji%ROWTYPE;
31    BEGIN
32        OPEN emp_par_dep_mounji(p_deptno);
33    LOOP
34        FETCH emp_par_dep_mounji INTO emp_rec;
35        EXIT WHEN emp_par_dep_mounji%NOTFOUND;
36        DBMS_OUTPUT.PUT_LINE('EmpNo:' || emp_rec.empno || ', Ename:' || emp_rec.ename);
37    END LOOP;
38    CLOSE emp_par_dep_mounji;
39    END afficher_emp_mounji;
40 END mounji_package;
41 /
```

3.1.3 Resultat :

```

SET SERVEROUTPUT ON;
CREATE OR REPLACE PACKAGE mounji_package IS
  PROCEDURE raisesalary_mounji (
    p_deptno NUMBER,
    p_emp_id IN NUMBER,
    p_amount IN NUMBER
  );
  PROCEDURE afficher_emp_mounji(p_deptno NUMBER);
END mounji_package;
/
Package cree.

```

FIGURE 3.1 – Création du package

```

SET SERVEROUTPUT ON;
SQL> CREATE OR REPLACE PACKAGE BODY mounji_package IS
 2   PROCEDURE raisesalary_mounji (
 3     p_deptno NUMBER,
 4     p_emp_id IN NUMBER,
 5     p_amount IN NUMBER
 6   )
 7   IS
 8     v_sal_courant emp.salaryTYPE;
 9     v_sal_nv      emp.salaryTYPE;
10     v_emp_job    emp.jobTYPE;
11   BEGIN
12     SELECT sal, job INTO v_sal_courant, v_emp_job
13     FROM emp
14     WHERE empno = p_emp_id;
15     v_sal_nv := v_sal_courant + p_amount;
16     IF salok_mounji(v_emp_job, v_sal_nv) = 1 THEN
17       UPDATE emp
18         SET sal = v_sal_nv, job = v_emp_job
19         WHERE empno = p_emp_id;
20     END IF;
21     DBMS_OUTPUT.PUT_LINE('Salaire augmente avec succes.');
22     COMMIT;
23   ELSE
24     DBMS_OUTPUT.PUT_LINE('Erreur : Le nouveau salaire est en dehors de l''intervalle autorise.');
25   END IF;
26   END raisesalary_mounji;
27
28  PROCEDURE afficher_emp_mounji(p_deptno NUMBER) IS
29    emp_rec emp_par_dep_mounji%ROWTYPE;
30  BEGIN
31    OPEN emp_par_dep_mounji(p_deptno);
32    LOOP
33      FETCH emp_par_dep_mounji INTO emp_rec;
34      EXIT WHEN emp_par_dep_mounji%NOTFOUND;
35      DBMS_OUTPUT.PUT_LINE('Emplo: ' || emp_rec.empno || ', Ename: ' || emp_rec.ename);
36    END LOOP;
37  CLOSE emp_par_dep_mounji;
38  END afficher_emp_mounji;
39 END mounji_package;
40 /

```

Corps de package cree.

FIGURE 3.2 – Création du Body du package

3.2 Question 1 :

3.2.1 Question :

Question

(Nom du trigger : raise_votrenom) Le salaire d'un employé ne diminue jamais

3.2.2 Code :

sql

```

1  CREATE OR REPLACE TRIGGER raise_mounji
2  BEFORE UPDATE OF sal ON emp
3  FOR EACH ROW
4  BEGIN
5    IF :NEW.sal < :OLD.sal THEN
6      :NEW.sal := :OLD.sal;
7    END IF;
8  END raise_mounji;
9  /
10
11 UPDATE emp SET sal = sal - 100 WHERE empno = 7000;
12
13 UPDATE emp SET sal = sal + 100 WHERE empno = 7000;

```

3.2.3 Resultat :

3.2.3.1 Code :

```
CREATE OR REPLACE TRIGGER raise_mounji
 2 BEFORE UPDATE OF sal ON emp
 3 FOR EACH ROW
 4 BEGIN
 5   IF :NEW.sal < :OLD.sal THEN
 6     :NEW.sal := :OLD.sal;
 7   END IF;
 8 END raise_mounji;
 9 /
```

Declencheur cree.

FIGURE 3.3 – Code du trigger

3.2.3.2 Test :

```
SQL> UPDATE emp SET sal = sal + 100 WHERE empno = 7000;
1 ligne mise a jour.

SQL> select empno , sal from emp;
      EMPNO        SAL
-----  -----
    7369       1000
    7499       1600
    7521       1250
    7566       2975
    7654       1250
    7698       2850
    7782       2450
    7788       3000
    7839       5000
    7844       1500
    7876       1100

      EMPNO        SAL
-----  -----
    7900        950
    7902       3000
    7934       1300
    7000       3500

15 lignes selectionnees.

SQL> UPDATE emp SET sal = sal - 100 WHERE empno = 7000;
1 ligne mise a jour.

SQL> select empno , sal from emp;
      EMPNO        SAL
-----  -----
    7369       1000
    7499       1600
    7521       1250
    7566       2975
    7654       1250
    7698       2850
    7782       2450
    7788       3000
    7839       5000
    7844       1500
    7876       1100

      EMPNO        SAL
-----  -----
    7900        950
    7902       3000
    7934       1300
    7000       3600

15 lignes selectionnees.
```

FIGURE 3.4 – Test 1

SQL>

FIGURE 3.5 – Test 2

3.3 Question 2 :

3.3.1 Question :

Question

(Nom du trigger : numdept_votrenom) Le numéro de département doit être entre 61 - 69.

3.3.2 Code :

```
sql
1 CREATE OR REPLACE TRIGGER numdept_mounji
2 BEFORE INSERT OR UPDATE OF deptno ON emp
3 FOR EACH ROW
4 BEGIN
5   IF :NEW.deptno IS NOT NULL AND (:NEW.deptno < 61 OR :NEW.deptno > 69) THEN
6     :NEW.deptno := NVL(:OLD.deptno, :NEW.deptno);
7   END IF;
8 END numdept_mounji;
9 /
10 -----
11 UPDATE emp SET deptno = 70 WHERE empno = 7000;
12 -----
13 INSERT INTO dept (deptno, dname, loc) VALUES (65, 'Devops', 'Chicago');
14 UPDATE emp SET deptno = 65 WHERE empno = 7000;
```

3.3.3 Resultat :

3.3.3.1 Code :

```
CREATE OR REPLACE TRIGGER numdept_mounji
2 BEFORE INSERT OR UPDATE OF deptno ON emp
3 FOR EACH ROW
4 BEGIN
5   IF :NEW.deptno IS NOT NULL AND (:NEW.deptno < 61 OR :NEW.deptno > 69) THEN
6     :NEW.deptno := NVL(:OLD.deptno, :NEW.deptno);
7   END IF;
8 END numdept_mounji;
9 /
```

Declencheur cree.

FIGURE 3.6 – Code du trigger

3.3.3.2 Test :

3.3.3.2.1 Ajouter département numéro 65 pour le Test :

```
SQL> select * from dept;  
  
DEPTNO DNAME          LOC  
-----  
      10 ACCOUNTING    NEW YORK  
      20 RESEARCH       DALLAS  
      30 SALES          CHICAGO  
      50 IT              New York  
  
SQL> INSERT INTO dept (deptno, dname, loc) VALUES (65, 'Devops', 'Chicago');  
1 ligne creee.  
  
SQL> commit;  
Validation effectuee.  
  
SQL> select * from dept;  
  
DEPTNO DNAME          LOC  
-----  
      10 ACCOUNTING    NEW YORK  
      20 RESEARCH       DALLAS  
      30 SALES          CHICAGO  
      50 IT              New York  
      65 Devops         Chicago
```

FIGURE 3.7 – Ajouter département numéro 65 pour le Test

3.3.3.2.2 Tests :

```
SQL> UPDATE emp SET deptno = 70 WHERE empno = 7000;  
1 ligne mise a jour.  
  
SQL> select empno , deptno from emp;  
  
EMPNO   DEPTNO  
-----  
 7369     20  
 7499     30  
 7521     30  
 7566     20  
 7654     30  
 7698     30  
 7782     10  
 7788     20  
 7839     10  
 7844     30  
 7876     20  
  
EMPNO   DEPTNO  
-----  
 7900     30  
 7902     20  
 7934     10  
 7000     20  
  
15 lignes selectionnees.  
  
SQL> UPDATE emp SET deptno = 65 WHERE empno = 7000;  
1 ligne mise a jour.  
  
SQL> select empno , deptno from emp;  
  
EMPNO   DEPTNO  
-----  
 7369     20  
 7499     30  
 7521     30  
 7566     20  
 7654     30  
 7698     30  
 7782     10  
 7788     20  
 7839     10  
 7844     30  
 7876     20  
  
EMPNO   DEPTNO  
-----  
 7900     30  
 7902     20  
 7934     10  
 7000     65  
  
15 lignes selectionnees.  
  
SQL> ■
```

FIGURE 3.8 – Test 1

FIGURE 3.9 – Test 2

3.4 Question 3 :

3.4.1 Question :

Question

(Nom du trigger : dept_votrenom) Si un employé est affecté à un département qui n'existe pas dans la base de données, ce département doit être rajouté avec pour valeur « A SAISIR » pour les attributs Dname et Loc.

3.4.2 Code :

```
sql

1 CREATE OR REPLACE TRIGGER dept_mounji
2 BEFORE INSERT OR UPDATE ON emp
3 FOR EACH ROW
4 DECLARE
5   v_department_exists NUMBER;
6   v_dname VARCHAR2(100);
7   v_loc VARCHAR2(100);
8 BEGIN
9   SELECT COUNT(*) INTO v_department_exists FROM dept WHERE deptno = :NEW.deptno;
10  IF v_department_exists = 0 THEN
11    v_dname := '&Enter_department_name';
12    v_loc := '&Enter_location';
13    INSERT INTO dept(deptno, dname, loc)
14      VALUES (:NEW.deptno, v_dname, v_loc);
15  END IF;
16 END dept_mounji;
17 /
18 -----
19 UPDATE emp SET deptno = 64 WHERE empno = 7000;
```

3.4.3 Resultat :

```
SQL> select * from dept;
DEPTNO DNAME          LOC
----- -----
 10 ACCOUNTING    NEW YORK
 20 RESEARCH      DALLAS
 30 SALES         CHICAGO
 50 IT            New York
 65 Devops        Chicago
-----
SQL> UPDATE emp SET deptno = 64 WHERE empno = 7000;
1 ligne mise a jour.

SQL> select * from dept;
DEPTNO DNAME          LOC
----- -----
 10 ACCOUNTING    NEW YORK
 20 RESEARCH      DALLAS
 30 SALES         CHICAGO
 50 IT            New York
 65 Devops        Chicago
 64 MARKETING    DALLAS
-----
6 lignes selectionnees.
```

FIGURE 3.10 – Code pour question 3

FIGURE 3.11 – Resultat pour question 3

3.5 Question 4 :

3.5.1 Question :

Question

(Nom du trigger : noweek_votrenom) Pour des raisons de sécurité, on souhaite interdire toute modification de la relation employé pendant le week-end (samedi et dimanche).

3.5.2 Code :

sql

```
1 CREATE OR REPLACE TRIGGER noweek_mounji
2 BEFORE INSERT OR UPDATE ON emp
3 FOR EACH ROW
4 DECLARE
5   v_day VARCHAR2(9);
6 BEGIN
7   SELECT TO_CHAR(SYSDATE, 'DAY') INTO v_day FROM dual;
8   IF v_day IN ('SATURDAY', 'SUNDAY') THEN
9     RAISE_APPLICATION_ERROR(-20001, 'Modifications not allowed on weekends.');
10  END IF;
11 END noweek_mounji;
12 /
```

3.5.3 Resultat :

```
CREATE OR REPLACE TRIGGER noweek_mounji
 2 BEFORE INSERT OR UPDATE ON emp
 3 FOR EACH ROW
 4 DECLARE
 5   v_day VARCHAR2(9);
 6 BEGIN
 7   SELECT TO_CHAR(SYSDATE, 'DAY') INTO v_day FROM dual;
 8   IF v_day IN ('SATURDAY', 'SUNDAY') THEN
 9     RAISE_APPLICATION_ERROR(-20001, 'Modifications not allowed on weekends.');
10  END IF;
11 END noweek_mounji;
12 /
```

Declencheur cree.

FIGURE 3.12 – Question 4

3.6 Question 5 :

3.6.1 Question :

Question

Désactiver le trigger de la question précédente et tester le.

3.6.2 Code :

```
sql  
1 ALTER TRIGGER noweek_mounji DISABLE;
```

3.6.3 Resultat :

SQL> ALTER TRIGGER noweek_mounji DISABLE;

Declencheur modifie.

FIGURE 3.13 – Question 5

3.7 Question 6 :

3.7.1 Question :

Question

Réactiver le trigger la question précédente.

3.7.2 Code :

```
sql  
1 ALTER TRIGGER noweek_mounji ENABLE;
```

3.7.3 Resultat :

SQL> ALTER TRIGGER noweek_mounji ENABLE;

Declencheur modifie.

FIGURE 3.14 – Question 6

3.8 Question 7 :

3.8.1 Question :

Question

Nom du déclencheur : `stat_votrenom`) On souhaite conserver des statistiques concernant les mises à jour (insertions, modifications, suppressions) sur la table `EMP`. Créez à l'aide de SQL la structure de la table `STATS_votrenom` et la peupler comme suit.

`STATS_votrenom (TypeMaj, NbMaj, Date_derniere_Maj)`

TypeMaj	NbMaj	Date_derniere_Maj
INSERT	0	NULL
UPDATE	0	NULL
DELETE	0	NULL

A) Définir un déclencheur après insertion/modification/suppression de la table `EMP` permettant de mettre à jour automatiquement la table `STATS_votrenom`. Tester son utilisation en effectuant diverses mises à jour sur la table `EMP`.

B) Tester l'effet de la présence et de l'absence de la clause `FOR EACH ROW` sur le comportement du déclencheur en utilisant une requête qui modifie plusieurs n-uplets (ex. `UPDATE EMP SET SAL = SAL * 1.05;`).

3.8.2 Création de la Table `STATS_mounji` :

sql

```
1 CREATE TABLE STATS_mounji (
2     TypeMaj VARCHAR2(10),
3     NbMaj NUMBER(5) DEFAULT 0,
4     Date_derniere_Maj DATE
5 );
6 -----
7 INSERT INTO STATS_mounji (TypeMaj, NbMaj, Date_derniere_Maj)
8 VALUES ('INSERT', 0, NULL);
9
10 INSERT INTO STATS_mounji (TypeMaj, NbMaj, Date_derniere_Maj)
11 VALUES ('UPDATE', 0, NULL);
12
13 INSERT INTO STATS_mounji (TypeMaj, NbMaj, Date_derniere_Maj)
14 VALUES ('DELETE', 0, NULL);
```

3.8.2.1 Résultat de la création de la table :

```
CREATE TABLE STATS_mounji (
    TypeMaj VARCHAR2(10),
    2      NbMaj NUMBER(5) DEFAULT 0,
    4      Date_derniere_Maj DATE
    5  );

Table creee.

INSERT INTO STATS_mounji (TypeMaj, NbMaj, Date_derniere_Maj)
2 VALUES ('INSERT', 0, NULL);

INSERT INTO STATS_mounji (TypeMaj, NbMaj, Date_derniere_Maj)
VALUES ('UPDATE', 0, NULL);

INSERT INTO STATS_mounji (TypeMaj, NbMaj, Date_derniere_Maj)

1 ligne creee.

SQL> SQL> 2
1 ligne creee.

SQL> SQL> 2 VALUES ('DELETE', 0, NULL);

1 ligne creee.

SQL> select * from stats_mounji;

TYPEMAJ          NBMAJ DATE_DER
-----  -----
INSERT            0
UPDATE            0
DELETE            0
```

FIGURE 3.15 – Crédit de la table STATS_mounji

3.8.3 Question A :

```

sql

1 CREATE OR REPLACE TRIGGER stat_mounji
2 AFTER INSERT OR UPDATE OR DELETE ON EMP
3 FOR EACH ROW
4 DECLARE
5   v_typemaj VARCHAR2(10);
6 BEGIN
7   IF INSERTING THEN
8     v_typemaj := 'INSERT';
9   ELSIF UPDATING THEN
10    v_typemaj := 'UPDATE';
11   ELSIF DELETING THEN
12    v_typemaj := 'DELETE';
13   END IF;
14
15   UPDATE STATS_mounji
16   SET NbMaj = NbMaj + 1,
17       Date_derniere_Maj = SYSDATE
18   WHERE TypeMaj = v_typemaj;
19 END stat_mounji;
20 /
21
22 INSERT INTO EMP VALUES (8000, 'John', 'Manager', NULL, SYSDATE, 3500, NULL, 10);
23 UPDATE EMP SET sal = 3500 WHERE empno = 7000;
24 DELETE FROM EMP WHERE empno = 8000;
25
26 SELECT * FROM STATS_mounji;

```

3.8.3.1 Resultat :

```

CREATE OR REPLACE TRIGGER stat_mounji
2 AFTER INSERT OR UPDATE OR DELETE ON EMP
3 FOR EACH ROW
4 DECLARE
5   v_typemaj VARCHAR2(10);
6 BEGIN
7   IF INSERTING THEN
8     v_typemaj := 'INSERT';
9   ELSIF UPDATING THEN
10    v_typemaj := 'UPDATE';
11   ELSIF DELETING THEN
12    v_typemaj := 'DELETE';
13   END IF;
14
15   UPDATE STATS_mounji
16   SET NbMaj = NbMaj + 1,
17       Date_derniere_Maj = SYSDATE
18   WHERE TypeMaj = v_typemaj;
19 END stat_mounji;
20 /

```

Declencheur cree.

FIGURE 3.16 – Code du trigger

```

SQL> 1 ligne mise a jour.
SQL> DELETE FROM EMP WHERE empno = 8000;
1 ligne supprimee.
SQL> SELECT * FROM STATS_mounji;
TYPEMAJ      NBMAJ DATE_DER
----- -----
INSERT        1 23/01/24
UPDATE        1 23/01/24
DELETE        1 23/01/24

SQL> INSERT INTO EMP VALUES (8000, 'John', 'Manager', NULL, SYSDATE, 3500, NULL, 10);
1 ligne creee.

SQL> SELECT * FROM STATS_mounji;
TYPEMAJ      NBMAJ DATE_DER
----- -----
INSERT        2 23/01/24
UPDATE        1 23/01/24
DELETE        1 23/01/24

SQL> DELETE FROM EMP WHERE empno = 8000;
1 ligne supprimee.

SQL> SELECT * FROM STATS_mounji;
TYPEMAJ      NBMAJ DATE_DER
----- -----
INSERT        2 23/01/24
UPDATE        1 23/01/24
DELETE        2 23/01/24

SQL> ■

```

FIGURE 3.17 – Resultat du Test du trigger

3.8.4 Question B :

3.8.4.1 Resultat :

```
CREATE OR REPLACE TRIGGER stat_mounji
2 AFTER INSERT OR UPDATE OR DELETE ON EMP
3 DECLARE
4   v_typemaj VARCHAR2(10);
5 BEGIN
6   IF INSERTING THEN
7     v_typemaj := 'INSERT';
8   ELSIF UPDATING THEN
9     v_typemaj := 'UPDATE';
10  ELSIF DELETING THEN
11    v_typemaj := 'DELETE';
12  END IF;
13
14  UPDATE STATS_mounji
15  SET NbMaj = NbMaj + 1,
16      Date_derniere_Maj = SYSDATE
17  WHERE TypeMaj = v_typemaj;
18 END stat_mounji;
19 /
```

SQL> UPDATE EMP SET SAL = SAL * 1.05;
15 lignes mises à jour.

SQL> SELECT * FROM STATS_mounji;

TYPEMAJ	NBMAJ	DATE_DER
INSERT	2	23/01/24
UPDATE	16	23/01/24
DELETE	2	23/01/24

Declencheur cree.

SQL> UPDATE EMP SET SAL = SAL * 1.05;
15 lignes mises à jour.

FIGURE 3.18 – Test Avec FOR EACH ROW

```
SQL> SELECT * FROM STATS_mounji;
```

TYPEMAJ	NBMAJ	DATE_DER
INSERT	2	23/01/24
UPDATE	17	23/01/24
DELETE	2	23/01/24

FIGURE 3.19 – Test Sans FOR EACH ROW

3.9 Question 8 :

3.9.1 Question :

Question

(Nom du déclencheur : `checksal_votrenom`) Chaque fois qu'un employé change de « job », on lui accorde une augmentation de salaire d'un montant de 100 euros. Attention ! Le président (avant le changement) n'est pas concerné car il est déjà bien payé... Le nouveau salaire doit être dans l'intervalle de salaires (entre `lSal` et `hSal` dans la table `SalIntervalle` du TP précédent) pour le nouveau job. Si le nouveau salaire est supérieur à la valeur de la colonne `hSal`, dans ce cas, il faudrait lui affecter la valeur de `hSal`, et si le nouveau salaire est inférieur à la valeur de `lSal`, affecter lui la valeur de `lSal`.

3.9.2 Code :

```
sql

1 CREATE OR REPLACE TRIGGER checksal_mounji
2 BEFORE UPDATE ON EMP
3 FOR EACH ROW
4 DECLARE
5   v_new_salary NUMBER;
6   v_lsal SalIntervalle_mounji.lsal%TYPE;
7   v_hsal SalIntervalle_mounji.hsal%TYPE;
8 BEGIN
9   IF :NEW.job != :OLD.job THEN
10    v_new_salary := :NEW.sal + 100;
11
12   IF :NEW.job != 'PRESIDENT' THEN
13     SELECT lsal, hsal INTO v_lsal, v_hsal
14     FROM SalIntervalle_mounji
15     WHERE job = :NEW.job;
16
17   IF v_new_salary > v_hsal THEN
18     :NEW.sal := v_hsal;
19   ELSIF v_new_salary < v_lsal THEN
20     :NEW.sal := v_lsal;
21   ELSE
22     mounji_package.raisesalary_mounji(:NEW.empno, 100);
23   END IF;
24   END IF;
25 END checksal_mounji;
26 /
27 -----
28 INSERT INTO EMP VALUES (8000, 'John', 'CLERK', NULL, SYSDATE, 1100, NULL, 10);
29 UPDATE EMP SET job = 'MANAGER' WHERE empno = 8000;
```

3.9.3 Resultat :

```

CREATE OR REPLACE TRIGGER checksal_mounji
  2 BEFORE UPDATE ON EMP
  3 FOR EACH ROW
  4 DECLARE
  5   v_new_salary NUMBER;
  6   v_lsal SalIntervalle_mounji.lsal%TYPE;
  7   v_hsal SalIntervalle_mounji.hsal%TYPE;
  8 BEGIN
  9   IF :NEW.job != :OLD.job THEN
10     v_new_salary := :NEW.sal + 100;
11
12   IF :NEW.job != 'PRESIDENT' THEN
13     SELECT lsal, hsal INTO v_lsal, v_hsal
14     FROM SalIntervalle_mounji
15     WHERE job = :NEW.job;
16
17   IF v_new_salary > v_hsal THEN
18     :NEW.sal := v_hsal;
19   ELSIF v_new_salary < v_lsal THEN
20     :NEW.sal := v_lsal;
21   ELSE
22     mounji_package.raisesalary_mounji(:NEW.empno, 100);
23   END IF;
24 END IF;
25 END IF;
26 END checksal_mounji;
27 /

```

Declencheur cree.

FIGURE 3.20 – Code du trigger

1 ligne creeee.

SQL> commit;

Validation effectuee.

SQL> select empno , job , sal from emp;

EMPNO	JOB	SAL
7369	CLERK	1102,5
7499	SALESMAN	1764
7521	SALESMAN	1378,13
7566	MANAGER	3279,94
7654	SALESMAN	1378,13
7698	MANAGER	3142,13
7782	MANAGER	2701,13
7788	ANALYST	3307,5
7839	PRESIDENT	5512,5
7844	SALESMAN	1653,75
7876	CLERK	1212,75

EMPNO	JOB	SAL
7900	CLERK	1047,38
7902	ANALYST	3307,5
7934	CLERK	1433,25
7000	SALESMAN	3969
8000	CLERK	1100

16 lignes selectionnees.

FIGURE 3.21 – Ajouter Un Clerk pour le Test

SQL> UPDATE EMP SET job = 'MANAGER' WHERE empno = 8000;

1 ligne mise a jour.

SQL> select empno , job , sal from emp;

EMPNO	JOB	SAL
7369	CLERK	1102,5
7499	SALESMAN	1764
7521	SALESMAN	1378,13
7566	MANAGER	3279,94
7654	SALESMAN	1378,13
7698	MANAGER	3142,13
7782	MANAGER	2701,13
7788	ANALYST	3307,5
7839	PRESIDENT	5512,5
7844	SALESMAN	1653,75
7876	CLERK	1212,75
7900	CLERK	1047,38
7902	ANALYST	3307,5
7934	CLERK	1433,25
7000	SALESMAN	3969
8000	MANAGER	2400

16 lignes selectionnees.

FIGURE 3.22 – Test