
# Implémentation des SGBDs

# Transactions

*Auteur :*
Nathanaël Bayle

23 Décembre 2020

# Table des matières

# Chapitre 1

# Énoncés

## 1.1 Exercice 1

1. Ouvrez deux sessions, Session 1 et Session 2, sous le même login.
2. Exécutez les commandes suivantes à partir de la Session 1 :
   — INSERT INTO EMP VALUES (7000, 'Petit Lion', 'SALESMAN', 7902, to_date('17-12-1980', 'dd-mm-yyyy'), 800, NULL, 20) ;
   — INSERT INTO EMP VALUES (7001, 'Chaussette', 'SALESMAN', 7698, to_date('20-2-1981', 'dd-mm-yyyy'), 1600, 300, 20) ;
3. Retournez à la Session 2 et vérifiez si vos modifications sont visibles à partir de cette session. Que constatez-vous ?
4. A partir de la Session 1, faites un COMMIT des modifications et vérifiez si les modifications sont connues dans la Session 2.
5. A partir de la Session 1, augmenter le salaire de l'employé 'petit lion' de 200 Euros.
6. A partir de la Session 2, mettez le salaire de l'employé 'petit lion' à 700 Euros. Que se passe-t-il ?
7. Faites un COMMIT dans la Session 1. Que se passe-t-il ? Faites un select dans les 2 sessions pour voir la modification.
8. Faites un COMMIT dans la deuxième session. Faites un select dans les 2 sessions pour voir la modification.
9. Utilisez un SELECT FOR UPDATE sur la Session 1 et essayez de modifier le salaire des employés à partir de la Session 2.

## 1.2 Exercice 2

**Rappel : mode de fonctionnement par défaut d'Oracle**

— Les lectures ne bloquent ni les autres lectures ni les écritures.
— Les lectures ne sont bloquées par rien, même pas par un blocage d'une table en mode exclusif.
— Il n'y pas de lecture impropre.
— Il n'y a pas de pertes de mises à jour.
— Il peut y avoir des lectures non reproductibles.
— Il peut y avoir des lignes fantômes.

1. Pour chacun des scénarios ci-dessous, exécuter la séquence d'instructions à partir de deux sessions (session 1 et session 2) et expliquer quels sont les phénomènes observés ?

   (a) **Scénario 1**

   **Session 1 :** select sal from emp where empno = 7369 ;

   **Session 2 :** update emp set sal= 802 where empno = 7369 ;

   **Session 2 :** commit ;

   **Session 1 :** select sal from emp where empno = 7369 ;

   (b) **Scénario 2**

   **Session 1 :** update emp set sal= 801 where empno = 7369 ;

   **Session 2 :** select sal from emp where empno = 7369 ;

   **Session 1 :** commit ;

   **Session 2 :** select sal from emp where empno = 7369 ;

   (c) **Scénario 3**

   **Session 1 :** update emp set sal= 803 where empno = 7369 ;

   **Session 2 :** update emp set sal= 804 where empno = 7369 ;

   **Session 1 :** commit ;

   **Session 2 :** commit ;

   (d) **Scénario 4**

   **Session 1 :** update emp set sal= 805 where empno = 7369 ;

   **Session 2 :** update emp set sal = 1300 where empno=7521 ;

   **Session 1 :** update emp set sal = 1300 where empno=7521 ;

   **Session 2 :** update emp set sal= 805 where empno = 7369 ;

   Attendre un moment et observer ce qui se passe.

   **Session 1 :** ROLLBACK ;

   Expliquer ce qui s'est passé.

## 1.3 Exercice 3

### A. Empêcher les lectures non reproductibles

Que pouvez-vous faire pour empêcher les lectures non reproductibles, dans chacun des cas suivants :

— dans le cas où la transaction ne modifie aucune donnée.

— dans le cas où elle modifie des données.

### B. Empêcher les lignes fantômes

Mêmes questions que l'exercice précédent, mais pour les lignes fantômes.

— **Read Only**

1. Ouvrez une nouvelle transaction en "READ ONLY".
2. Ouvrez en parallèle une deuxième transaction dans laquelle vous modifiez les salaires des employés.
3. Validez cette deuxième transaction. Voyez-vous les modifications dans la première transaction ?
4. Essayez de modifier des données dans la première transaction.
5. Que se serait-il passé si la première transaction n'avait pas été en "READ ONLY" ? Vérifiez-le après avoir terminé la première transaction.

— **Mode serialisable**

Essayez de faire cet exercice en devinant ce qui va se passer avant de lancer chaque commande.

1. Ouvrez 2 sessions de travail avec des transactions T1 et T2.
2. Passez T1 en mode sérialisé.
3. T1 affiche tous les noms et salaires des employés.
4. T2 modifie le salaire de l'employé 'WARD'.
5. T1 affiche à nouveau tous les salaires. Quel salaire voit-il pour l'employé 'WARD' ? Pourquoi ?
6. T2 valide sa transaction.
7. T1 affiche à nouveau tous les salaires. Quel salaire voit-il pour l'employé 'WARD' ? Pourquoi ?
8. T1 modifie le salaire de l'employé 'WARD'. Que se passe-t-il ? Pourquoi ?
9. T1 modifie le salaire d'un autre employé. Que se passe-t-il ?

   Voyez-vous une différence avec le mode par défaut d'Oracle ? Explications ?

# Chapitre 2

# Résolutions

## 2.1 Exercice 1

Dans un premier temps on ouvre les deux sessions dans deux terminal avec le même login.

```
[nabayle1@turing ~]$ ssh nabayle1@tp-l3bd-nabayle1.local.isima.fr
nabayle1@tp-l3bd-nabayle1.local.isima.fr's password:
Last login: Mon Dec 14 16:19:13 2020 from turing.local.isima.fr
[nabayle1@tp-l3bd-nabayle1 ~]$ sudo su oracle
[oracle@tp-l3bd-nabayle1 nabayle1]$ rlwrap sqlplus / as sysdba

SQL*Plus: Release 18.0.0.0.0 - Production on Mon Dec 14 16:20:55 2020
Version 18.3.0.0.0

Copyright (c) 1982, 2018, Oracle.  All rights reserved.


Connected to:
Oracle Database 18c Enterprise Edition Release 18.0.0.0.0 - Production
Version 18.3.0.0.0

SQL>
```

Depuis la première session, on exécute les commandes suivantes :

- INSERT INTO EMP VALUES (7000, 'Petit Lion', 'SALESMAN',7902, to_date( '17-12-1980', 'dd-mm-yyyy'), 800, NULL, 20) ;
- INSERT INTO EMP VALUES (7001, 'Chaussette', 'SALESMAN', 7698, to_date( '20-2-1981', 'dd-mm-yyyy'), 1600, 300, 20) ;

On se rend compte que dans la deuxième session, les changements apportés ne sont pas visibles. Si depuis la première session, on effectue un **COMMIT**, alors les modifications serons visibles depuis la deuxième session.

A partir de la Session 1, on exécute cette commande :

**UPDATE** emp **SET** sal = sal + 200 **WHERE** ename = 'Petit Lion' ;

Puis depuis la Session 2, on exécute cette commande :

**UPDATE** emp **SET** sal = 700 **WHERE** ename = 'Petit Lion' ;

On remarque que la Session 2 est bloqué en attendant que la Session 1 effectue un **COMMIT**. Une fois le **COMMIT** effectué, les données sont mis à jours. On se rend compte que les modifications sont enregistré pour chaque Sessions.

**Session 1 :** `7000 Petit Lion SALESMAN 7902 17-DEC-80 1000`

**Session 2 :** `7000 Petit Lion SALESMAN 7902 17-DEC-80 700`

Lorsque l'on fait un **COMMIT** sur la deuxième session, la valeur est alors modifié pour tout le monde.

**Session 1 :** `7000 Petit Lion SALESMAN 7902 17-DEC-80 700`

**Session 2 :** `7000 Petit Lion SALESMAN 7902 17-DEC-80 700`

## 2.2 Exercice 2

### 2.2.1 Scénario 1

```
SQL> select sal from emp where empno = 7369 ;

       SAL
----------
       800
```

Figure 2.1 Session 1

On sélectionne sal de la table emp ou empno = 7369.

```
SQL> update emp set sal= 802 where empno = 7369 ;

1 row updated.
```

Figure 2.2 Session 2

On update le sal du empno = 7369.

```
SQL> commit ;

Commit complete.
```

Figure 2.3 Session 2

Puis on fait un commit ;

```
SQL> select sal from emp where empno = 7369 ;

       SAL
----------
       802
```

Figure 2.4 Session 1

On se rend compte que les données sont mis à jour.

### 2.2.2 Scénario 2

```
SQL> update emp set sal= 801 where empno = 7369 ;

1 row updated.
```

Figure 2.5 Session 1

On met a jour le sal du empno = 7369.

```
SQL> select sal from emp where empno = 7369 ;

       SAL
----------
       802
```

Figure 2.6 Session 2

Depuis la session 2, les modifications ne sont pas visibles.

```
SQL> commit ;

Commit complete.
```

Figure 2.7 Session 1

On fait un commit.

```
SQL> select sal from emp where empno = 7369 ;

       SAL
----------
       801
```

Figure 2.8 Session 2

On se rend compte que les données sont mis à jour.

### 2.2.3 Scénario 3

```
SQL> update emp set sal= 803 where empno = 7369 ;

1 row updated.
```

Figure 2.9 Session 1

On met a jour le sal de empno = 7369.

```
SQL> update emp set sal= 804 where empno = 7369 ;
```

Figure 2.10 Session 2

On veut aussi mettre a jour le sal de empno = 7369, mais la session 2 est bloquée.

```
SQL> commit;

Commit complete.
```

Figure 2.11 Session 1

On fait un commit.

```
SQL> update emp set sal= 804 where empno = 7369 ;

1 row updated.

SQL>
```

Figure 2.12 Session 2

La session 2 est débloqué et met a jour le sal de empno = 7369.

```
SQL> commit;

Commit complete.
```

Figure 2.13 Session 2

On fait un commit.

### 2.2.4 Scénario 4

```
SQL> update emp set sal= 805 where empno = 7369 ;

1 row updated.
```

Figure 2.14 Session 1

On met a jour le sal de empno = 7369.

```
SQL> update emp set sal = 1300 where empno=7521 ;

1 row updated.
```

Figure 2.15 Session 2

On met a jour le sal de empno = 7521.

```
SQL> update emp set sal = 1300 where empno=7521 ;
```

Figure 2.16 Session 1

On veut aussi mettre a jour le sal de empno = 7521, mais la session 1 est bloquée car la session 2 n'as pas fait de commit.

```
SQL> update emp set sal= 805 where empno = 7369 ;
```

Figure 2.17 Session 2

On veut aussi mettre a jour le sal de empno = 7369, mais la session 2 est bloquée car la session 1 n'as pas fait de commit.

```
SQL> ROLLBACK ;

Rollback complete.
```

Figure 2.18 Session 1

On effectue un rollback sur la session 1 et on revient à la situation initiale.

```
SQL> update emp set sal= 805 where empno = 7369 ;

1 row updated.
```

Figure 2.19 Session 2

La mise à jour de la session 2 est alors validée.

## 2.3 Exercice 3

**A. Empêcher les lectures non reproductibles**

C'est une transaction qui relit des données qu'elle a lu précédemment et trouve que les données ont été modifiées par une autre transaction.

- Dans le cas où la transaction ne modifie aucune donnée, on met la transaction en mode **read only** car elle ne voit que les modifications validées avant son début. Une telle transaction ne peut pas modifier la base et elle n'est jamais bloquée.
- Dans le cas où elle modifie des données, on met la transaction en mode **serializable**. Ce niveau est comme **read only**, mais en plus la transaction $T$ peut modifier des n-uplets. Si $T$ tente de modifier (**delete** ou **update**) un n-uplet dont la dernière version a été produite par une autre transaction validée après le démarrage de $T$, Oracle considère qu'il y a un problème potentiel de sérialisabilité et provoque donc l'annulation de l'instruction DML et la propagation d'une erreur Oracle.

**B. Empêcher les lignes fantômes**

C'est une transaction qui ré-exécute une requête renvoyant un ensemble de lignes satisfaisant une condition de recherche et trouve que l'ensemble des lignes satisfaisant la condition a changé du fait d'une autre transaction récemment validée.

- Dans le cas où la transaction ne modifie aucune donnée, on peut passer la transaction en mode READ ONLY.
- Dans le cas où elle modifie des données, on peut passer en mode serializable ou bloquer les tables pour lesquels on ne veut pas de lignes fantômes.

### 2.3.1 Read Only

```
SQL> set transaction read only;

Transaction set.
```

Figure 2.20 Session 1

On ouvre une nouvelle transaction en "READ ONLY".

```
SQL> select * from emp where empno=7000;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------

      7000 Petit Lion SALESMAN        7902 17-DEC-80        700
        20
```

Figure 2.21 Session 1

```
SQL> update  emp set sal = 10000 where empno= 7000;

1 row updated.

SQL> select * from emp where empno=7000;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7000 Petit Lion SALESMAN        7902 17-DEC-80      10000
        20


SQL> commit;

Commit complete.
```

Figure 2.22 Session 2

On modifie le sal de empno=7000.

```
SQL> select * from emp where empno=7000;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------

      7000 Petit Lion SALESMAN        7902 17-DEC-80        700
        20
```

Figure 2.23 Session 1

On ce rend compte que le salaire n'as pas bougé.

Comme la session 1 est en read only, elle ne peut pas modifier les données.

Si la première transaction n'avait pas été en "READ ONLY", le nouveau salaire aurait été connu de la première transaction dès le commit de la deuxième transaction. On le vérifie après avoir tapé un COMMIT de la première transaction.

### 2.3.2 Mode serialisable

```
SQL> set transaction isolation level serializable;

Transaction set.

SQL> select * from emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7369 SMITH      CLERK           7902 17-DEC-80        800
        20

      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300
        30

      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500
        30
```

Figure 2.24 Session 1

On passe T1 en mode serialisable et on affiche tout depuis la table emp.

```
SQL> update emp set sal=10000 where ename='WARD';

1 row updated.
```

Figure 2.25 Session 2

T2 modifie le salaire de l'employé 'WARD'

```
SQL> select * from emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7369 SMITH      CLERK           7902 17-DEC-80        800
        20

      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300
        30

      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500
        30
```

Figure 2.26 Session 1

On affiche depuis T1, il n'y a pas de modification car T2 n'as pas effectué de commit.

```
SQL> commit;

Commit complete.
```

Figure 2.27 Session 2

T2 fait un commit.

```
SQL> select * from emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7369 SMITH      CLERK           7902 17-DEC-80        800
        20

      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300
        30

      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500
        30
```

Figure 2.28 Session 1

On affiche depuis T1, mais il n'y a toujours aucun changement. Ceci est expliqué grâce au faite que T1 soit en mode d'isolation sérialisable.

```
SQL> update emp set sal = 666 where ename='WARD';
update emp set sal = 666 where ename='WARD'
       *
ERROR at line 1:
ORA-08177: can't serialize access for this transaction
```

Figure 2.29 Session 1

On essaie de modifier le sal de 'WARD' mais on obtient une erreur car la transaction est validé pas T2 et donc sérialisable.

```
SQL> update emp set sal = 666 where empno = 7000 ;

1 row updated.
```

Figure 2.30 Session 1

Enfin on modifie le sal d'un autre emp et tout ce passe bien car cette transaction ne pause pas de problème au niveau d'isolation.

Dans le mode par défaut d'Oracle, T1 aurait vu le nouveau salaire de WARD après la validation de T2 et il aurait pu modifier lui aussi le salaire de WARD.

# Chapitre 3

# Annexe

```
CREATE TABLE DEPT
        (DEPTNO NUMBER(2) CONSTRAINT PK_DEPT PRIMARY KEY,
        DNAME VARCHAR2(14) ,
        LOC VARCHAR2(13));

CREATE TABLE EMP
        (EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
        ENAME VARCHAR2(10),
        JOB VARCHAR2(9),
        MGR NUMBER(4),
        HIREDATE DATE,
        SAL NUMBER(7,2),
        COMM NUMBER(7,2),
        DEPTNO NUMBER(2) CONSTRAINT FK_DEPTNO REFERENCES DEPT);

CREATE TABLE SALS
        (job varchar2(9),
        lsal number(7,2),
        hsal number(7,2));

INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW␣YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
INSERT INTO DEPT VALUES (30,'SALES','CHICAGO');
INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON');

INSERT INTO EMP VALUES (7369,'SMITH','CLERK',7902,to_date('17-12-1980','
    dd-mm-yyyy'),800,NULL,20);
INSERT INTO EMP VALUES (7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981'
    ,'dd-mm-yyyy'),1600,300,30);
INSERT INTO EMP VALUES (7521,'WARD','SALESMAN',7698,to_date('22-2-1981',
    'dd-mm-yyyy'),1250,500,30);
INSERT INTO EMP VALUES (7566,'JONES','MANAGER',7839,to_date('2-4-1981','
    dd-mm-yyyy'),2975,NULL,20);
INSERT INTO EMP VALUES (7654,'MARTIN','SALESMAN',7698,to_date('28-9-1981
    ','dd-mm-yyyy'),1250,1400,30);
INSERT INTO EMP VALUES (7698,'BLAKE','MANAGER',7839,to_date('1-5-1981','
    dd-mm-yyyy'),2850,NULL,30);
INSERT INTO EMP VALUES (7782,'CLARK','MANAGER',7839,to_date('9-6-1981','
    dd-mm-yyyy'),2450,NULL,10);
INSERT INTO EMP VALUES (7788,'SCOTT','ANALYST',7566,to_date('13-7-87','
    dd-mm-yy')-85,3000,NULL,20);
INSERT INTO EMP VALUES (7839,'KING','PRESIDENT',NULL,to_date('17-11-1981
    ','dd-mm-yyyy'),5000,NULL,10);
INSERT INTO EMP VALUES (7844,'TURNER','SALESMAN',7698,to_date('8-9-1981'
    ,'dd-mm-yyyy'),1500,0,30);
INSERT INTO EMP VALUES (7876,'ADAMS','CLERK',7788,to_date('13-7-87', 'dd
    -mm-yy')-51,1100,NULL,20);
INSERT INTO EMP VALUES (7900,'JAMES','CLERK',7698,to_date('3-12-1981','
    dd-mm-yyyy'),950,NULL,30);
INSERT INTO EMP VALUES (7902,'FORD','ANALYST',7566,to_date('3-12-1981','
    dd-mm-yyyy'),3000,NULL,20);
INSERT INTO EMP VALUES (7934,'MILLER','CLERK',7782,to_date('23-1-1982','
    dd-mm-yyyy'),1300,NULL,10);
```

```
INSERT INTO SALS VALUES ('ANALYST', 2500, 3000) ;
INSERT INTO SALS VALUES ('CLERK', 900, 1300) ;
INSERT INTO SALS VALUES ('MANAGER', 2400, 3000) ;
INSERT INTO SALS VALUES ('PRESIDENT', 4500, 4900) ;
INSERT INTO SALS VALUES ('SALESMAN', 1200, 1700) ;

COMMIT;
```