

TP5 BDA : Transactions et contrôle de concurrence

Exercice 1:

```
Invite de commandes - sqlplus user1/mdp1
Microsoft Windows [version 10.0.19045.5608]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\sadde>sqlplus user1/mdp1

SQL*Plus: Release 21.0.0.0.0 - Production on Mar. Avr. 15 14:37:42 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

Heure de la dernière connexion réussie : Mar. Avr. 15 2025 14:37:12 +02:00

Connecté à :
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL>
```

Tout d'abord, on modifie l'AUTO COMMIT à OFF afin que les modifications effectuées dans une session ne soient pas automatiquement validées (committed). Ceci va être réalisé pour les deux sessions.

1/ On crée la table Transaction au niveau de S1.

```
Invite de commandes - sqlplus / as sysdba;
C:\Users\sadde>sqlplus / as sysdba;

SQL*Plus: Release 21.0.0.0.0 - Production on Mar. Avr. 15 13:54:25 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

Connecté à :
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> SET AUTOCOMMIT OFF;
SQL> CREATE TABLE transaction(
  2 idTransaction VARCHAR2(44),
  3 valTransaction NUMBER(10));
Table créée.

SQL>
```

2/ On ajoute des lignes dans la table transaction, on modifie une ligne et on supprime une autre..

Invite de commandes - sqlplus / as sysdba;

C:\Users\sadde>sqlplus / as sysdba;

SQL*Plus: Release 21.0.0.0.0 - Production on Mar. Avr. 15 13:53:12 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

Connecté à :

Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

```
SQL> SET AUTOCOMMIT OFF;
SQL> INSERT INTO transaction VALUES('T1',100);

1 ligne créée.

SQL> INSERT INTO transaction VALUES('T2',200);

1 ligne créée.

SQL> INSERT INTO transaction VALUES('T3',300);

1 ligne créée.

SQL> UPDATE transaction SET valTransaction = 999 WHERE idTransaction = 'T1';

1 ligne mise à jour.

SQL> DELETE FROM transaction WHERE idTransaction = 'T2';

1 ligne supprimée.

SQL> SELECT * FROM transaction;

IDTRANSACTION                                VALTRANSACTION
-----
T1                                                    999
T3                                                    300

SQL> ROLLBACK;

Annulation (rollback) effectuée.

SQL> SELECT * FROM transaction;

aucune ligne sélectionnée
```

On procède à ce niveau à plusieurs opérations sur la table transaction: insertion de nouvelles lignes, modification d'une ligne existante, puis suppression d'une autre. Après cela, en exécutant un ROLLBACK, toutes les opérations réalisées sont annulées, ce qui signifie que la base de données revient à l'état initial avant la transaction.

```
SQL> SELECT * FROM transaction;
```

IDTRANSACTION	VALTRANSACTION
T1	999
T3	300

```
SQL> ROLLBACK;
```

Annulation (rollback) effectuée.

```
SQL> SELECT * FROM transaction;
```

Aucune ligne sélectionnée

```
SQL> INSERT INTO transaction VALUES('T1',100);
```

1 ligne créée.

```
SQL> INSERT INTO transaction VALUES('T2',200);
```

1 ligne créée.

```
SQL> INSERT INTO transaction VALUES('T3',300);
```

1 ligne créée.

```
SQL> quit
```

Se déconnecter de Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

```
C:\Users\sadde>
```

On insère des lignes, ensuite on quitte la session S2.

Par la suite, au niveau de la session S1. On lit le contenu de la table transaction. On remarque que malgré le fait qu'il n'y ait pas eu de COMMIT sur les données pour S2 avant de quitter la session. Les données restent visibles pour S1 étant donné que les deux sessions utilisent le même utilisateur donc même si on quitte la session S2, les données resteront visibles dans la session S1.

En fermant brutalement l'invite de commandes et en nous reconnectant, les modifications sont perdues vu que l'on n'a pas fait de COMMIT.

```
C:\Users\sadde>sqlplus / as sysdba

Microsoft Windows [version 10.0.19045.5608]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\sadde>sqlplus / as sysdba

SQL*Plus: Release 21.0.0.0.0 - Production on Mar. Avr. 15 15:21:08 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

Connecté à :
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> SELECT * FROM Transaction
  2  ;
```

IDTRANSACTION	VALTRANSACTION
T1	100
T2	200
T3	300

```
SQL> _
```

5/En ajoutant de nouvelles lignes à la table (Transactions T6 et T7) et en modifiant la table. En faisant un ROLLBACK, on remarque que les modifications réalisées ne sont pas annulées. En effet, Oracle fait un COMMIT automatique lorsqu'on modifie la table. Par conséquent, le ROLLBACK n'aura aucun effet.

```
C:\Users\sadde>sqlplus / as sysdba

Version 21.3.0.0.0

SQL> SET AUTOCOMMIT OFF;
SQL> INSERT INTO Transaction VALUES ('T6',600);

1 ligne créée.

SQL> INSERT INTO Transaction VALUES ('T7',700);

1 ligne créée.

SQL> ALTER TABLE transaction ADD val2transaction NUMBER(10);

Table modifiée.

SQL> ROLLBACK;

Annulation (rollback) effectuée.

SQL> SELECT * FROM TRANSACTION;
```

IDTRANSACTION	VALTRANSACTION	VAL2TRANSACTION
T6	600	
T7	700	
T1	100	
T2	200	
T3	300	

```
SQL> _
```

6/

Une session est une connexion ouverte entre un utilisateur et la base de données. Elle permet d'exécuter des requêtes SQL, d'effectuer des transactions, et reste active jusqu'à ce qu'on la ferme.

Une transaction est un ensemble cohérent d'opérations SQL exécutées entre deux points : le début de la transaction et sa fin, marquée par un COMMIT pour la validation ou un ROLLBACK pour l'annulation.

Exercice 2:

On crée les deux tables Client et Vol.

Invite de commandes - sqlplus / as sysdba

```
C:\Users\sadde>sqlplus / as sysdba

SQL*Plus: Release 21.0.0.0.0 - Production on Mar. Avr. 15 15:41:40 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

Connecté à :
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> CREATE TABLE client (
  2     idClient VARCHAR2(44),
  3     prenomClient VARCHAR2(11),
  4     nbrPlacesReserveesClient NUMBER(10)
  5 );

Table créée.

SQL> CREATE TABLE vol (
  2     idVol VARCHAR2(44),
  3     capaciteVol NUMBER(10),
  4     nbrPlacesReserveesVol NUMBER(10)
  5 );

Table créée.

SQL>
```

On va à ce niveau ajouter un vol et deux clients.

```
SQL> INSERT INTO vol (idVol, capaciteVol, nbrPlacesReserveesVol)
  2  VALUES ('V1',150,0);

1 ligne créée.

SQL> INSERT INTO client (idClient, prenomClient, nbrPlacesReserveesClient) VALUES ('C1','Elyes',0);

1 ligne créée.

SQL> INSERT INTO client(idClient,prenomClient,nbrPlacesReserveesClient) VALUES ('C2','John',0);

1 ligne créée.

SQL> COMMIT;

Validation effectuée.
```

Isolation des transactions

Au niveau de la session S1, on modifie le nombre de places réservées au niveau de la table Vol pour le vol V1. On modifie notamment le nombre de places réservées par le client C1. Ensuite, on fait un select, on remarque que les données sont visibles au niveau de cette session

```

SQL> INSERT INTO vol (idVol, capaciteVol, nbrPlacesReserveesVol)
  2 VALUES ('V1',150,0);

1 ligne cr   e.

SQL> INSERT INTO client (idClient, prenomClient, nbrPlacesReserveesClient) VALUES ('C1','Elyes',0);

1 ligne cr   e.

SQL> INSERT INTO client(idClient,prenomClient,nbrPlacesReserveesClient) VALUES ('C2','John',0);

1 ligne cr   e.

SQL> COMMIT;

Validation effectu  e.

SQL> UPDATE vol
  2 SET nbrPlacesReserveesVol = nbrPlacesReserveesVol + 1
  3 WHERE idVol = 'V1';

1 ligne mise    jour.

SQL> UPDATE client
  2 SET nbrPlacesReserveesClient = nbrPlacesReserveesClient + 1
  3 WHERE idClient = 'C1';

1 ligne mise    jour.

SQL> SELECT * FROM Vol;

IDVOL                                CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1                                150                1

SQL> SELECT * FROM CLIENT;

IDCLIENT                                PRENOMCLIEN
-----
NBRPLACESRESERVEESCLIENT
-----
C1                                Elyes
                                1
C2                                John
                                0

```

En tentant d'acc  der aux m  mes donn  es    partir de la session S2, on remarque qu'elles ne sont pas visibles. Ceci est d      l'isolation READ_COMMITTED qui ne permet que la lecture des donn  es qui ont   t   le sujet d'un COMMIT. Ainsi, s'il y a des modifications r  alis  es par une transaction A. Ces modifications ne seront visibles pour une transaction B que si un COMMIT a   t   r  alis   pour la transaction A.

```
Invite de commandes - sqlplus / as sysdba

C:\Users\sadde>sqlplus / as sysdba

SQL*Plus: Release 21.0.0.0.0 - Production on Mar. Avr. 15 15:46:26 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

Connecté à :
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED; On choisit le niveau d'isolation par défaut
Session modifiée.

SQL> SELECT * FROM CLIENT;

IDCLIENT          PRENOMCLIEN
-----
NBRPLACESRESERVEESCLIENT
-----
C1                0                Elyes
C2                0                John

SQL> SELECT * FROM VOL;

IDVOL          CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1            150                0

SQL>
```

COMMIT et ROLLBACK:

En effectuant un ROLLBACK, on revient à l'état initial de la table avant les changements.

```
SQL> SELECT * FROM Vol;

IDVOL          CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1            150                1

SQL> SELECT * FROM CLIENT;

IDCLIENT          PRENOMCLIEN
-----
NBRPLACESRESERVEESCLIENT
-----
C1                1                Elyes
C2                0                John

SQL> ROLLBACK;
Annulation (rollback) effectuée.

SQL> SELECT * FROM Vol;

IDVOL          CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1            150                0

SQL> SELECT * FROM Client;

IDCLIENT          PRENOMCLIEN
-----
NBRPLACESRESERVEESCLIENT
-----
C1                0                Elyes
C2                0                John
```


Ainsi, en effectuant un COMMIT à ce niveau, il n'y aura aucune modification étant donné qu'il s'agit d'une nouvelle transaction.

```
SQL> COMMIT;

Validation effectuée.

SQL> SELECT * FROM Vol;

IDVOL                                CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1                                150                0

SQL> SELECT* FROM CLIENT;

IDCLIENT                                PRENOMCLIEN
-----
NBRPLACESRESERVEESCLIENT
-----
C1                                Elyes
                                0
C2                                John
                                0
```

À ce niveau, on va refaire les modifications et valider par un COMMIT au niveau de S1.

```
SQL> UPDATE vol
  2 SET nbrPlacesReserveesVol = nbrPlacesReserveesVol + 1
  3 WHERE idVol = 'V1';

1 ligne mise à jour.

SQL> UPDATE client
  2 SET nbrPlacesReserveesClient = nbrPlacesReserveesClient + 1
  3 WHERE idClient = 'C1';

1 ligne mise à jour.

SQL> COMMIT;

Validation effectuée.

SQL> SELECT * FROM Vol;

IDVOL                                CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1                                150                1

SQL> SELECT * FROM CLIENT;

IDCLIENT                                PRENOMCLIEN
-----
NBRPLACESRESERVEESCLIENT
-----
C1                                Elyes
                                1
C2                                John
                                0
```

On remarque à ce niveau que l'utilisation d'un ROLLBACK n'annule pas les modifications.

```
SQL> COMMIT;
```

Validation effectuée.

```
SQL> SELECT * FROM Vol;
```

IDVOL	CAPACITEVOL	NBRPLACESRESERVEESVOL
V1	150	1

```
SQL> SELECT * FROM CLIENT;
```

IDCLIENT	NBRPLACESRESERVEESCLIENT	PRENOMCLIEN
C1	1	Elyes
C2	0	John

```
SQL> ROLLBACK;
```

Annulation (rollback) effectuée.

```
SQL> SELECT * FROM Vol;
```

IDVOL	CAPACITEVOL	NBRPLACESRESERVEESVOL
V1	150	1

```
SQL> SELECT * FROM Client;
```

IDCLIENT	NBRPLACESRESERVEESCLIENT	PRENOMCLIEN
C1	1	Elyes
C2	0	John

```
SQL> SELECT * FROM Vol;
```

IDVOL	CAPACITEVOL	NBRPLACESRESERVEESVOL
V1	150	0

```
SQL> SELECT* FROM CLIENT;
```

IDCLIENT	NBRPLACESRESERVEESCLIENT	PRENOMCLIEN
C1	0	Elyes
C2	0	John

Avant le COMMIT

```
SQL> SELECT * FROM Client;
```

IDCLIENT	NBRPLACESRESERVEESCLIENT	PRENOMCLIEN
C1	1	Elyes
C2	0	John

Après le COMMIT

```
SQL> SELECT * FROM Vol;
```

IDVOL	CAPACITEVOL	NBRPLACESRESERVEESVOL
V1	150	1

isolation incomplète = incohérence possible:

On retourne tout d'abord à l'état initial.

```
SQL> UPDATE client
  2 SET nbrPlacesReserveesClient = nbrPlacesReserveesClient - 1
  3 WHERE idClient = 'C1';

1 ligne mise à jour.

SQL> UPDATE vol
  2 SET nbrPlacesReserveesVol = nbrPlacesReserveesVol - 1
  3 WHERE idVol = 'V1';

1 ligne mise à jour.

SQL> SELECT * FROM Vol;
```

IDVOL	CAPACITEVOL	NBRPLACESRESERVEESVOL
V1	150	0

```
SQL> SELECT * FROM Client;
```

IDCLIENT	PRENOMCLIEN
C1	Elyes
C2	John

```
SQL> COMMIT;
```

Validation effectuée.

On effectue à ce niveau une réservation de deux billets pour C1 pour T1(Session S1).

```
SQL> UPDATE vol
    2 SET nbrPlacesReserveesVol = nbrPlacesReserveesVol + 2
    3 WHERE idVol = 'V1';

1 ligne mise à jour.
```

```
SQL> UPDATE client
    2 SET nbrPlacesReserveesClient = nbrPlacesReserveesClient + 2
    3 WHERE idClient = 'C1';

1 ligne mise à jour.
```

```
SQL> SELECT * FROM vol;
```

IDVOL	CAPACITEVOL	NBRPLACESRESERVEESVOL
V1	150	2

```
SQL> SELECT * FROM Client;
```

IDCLIENT	PrenomClient	NBRPLACESRESERVEESCIENT
C1	Elyes	2
C2	John	0

Par la suite, on effectue une réservation de trois billets pour C2 pour T2(Session S2). On notera que les modifications de T1 ne sont pas encore visibles vu qu'il n'y a pas encore eu de COMMIT.

```
SQL> UPDATE vol
  2 SET nbrPlacesReserveesVol = nbrPlacesReserveesVol + 3
  3 WHERE idVol = 'V1';
1 ligne mise à jour.

SQL>
SQL> UPDATE client
  2 SET nbrPlacesReserveesClient = nbrPlacesReserveesClient + 3
  3 WHERE idClient = 'C2';
1 ligne mise à jour.
```

On fait un COMMIT pour T1 et pour T2 respectivement. En lisant les données à partir des tables Vol et Client. On remarque que le client C1 a deux places enregistrées. Le client C2 a trois places enregistrées. Mais, seulement 3 de ces places sont présentes au niveau de la table Vol.

```
SQL> SELECT * FROM Client;

IDCLIENT          PRENOMCLIENT
-----
NBRPLACESRESERVEESCLIENT
-----
C1                2                Elyes
C2                3                John

SQL> SELECT * FROM Vol;

IDVOL          CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1            150                3
```

La base passe ainsi à un état incohérent. Il existe 5 places qui ont été réservées mais seulement 3 d'entre elles figurent dans la table Vol.

isolation complète = blocage et rejet des transactions possibles

Tout d'abord, on réinitialise la base de données à son état précédent

```
SQL> UPDATE vol
  2 SET nbrPlacesReserveesVol = 0
  3 WHERE idVol = 'V1';
1 ligne mise à jour.

SQL> UPDATE client
  2 SET nbrPlacesReserveesClient = 0
  3 ;
2 lignes mises à jour.

SQL> COMMIT;

Validation effectuée.

SQL> _
```

On doit utiliser le mode SERIALIZABLE pour les deux transactions.

```
SQL> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
Session modifiée.
```

On va maintenant reproduire l'exemple r1(d)w2(d)w2(d')C2w1(d')C1.

On commence par une lecture des données du vol par la transaction T1(Session 1).

```
SQL> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
Session modifiée.
SQL> SELECT * FROM Vol;

IDVOL                                CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1                                    150          0
```

Ensuite, le client 2 va réserver 3 places dans T2. Il devra donc écrire dans la table vol(w2(d)) et écrire dans la table Client (w2(d')) ensuite on valide les modifications.

```
SQL> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
Session modifiée.
SQL> UPDATE vol
  2 SET nbrPlacesReserveesVol = nbrPlacesReserveesVol + 2
  3 WHERE idVol = 'V1';

1 ligne mise à jour.
SQL> UPDATE client
  2 SET nbrPlacesReserveesClient = nbrPlacesReserveesClient + 3
  3 WHERE idClient = 'C2';

1 ligne mise à jour.
SQL> COMMIT;

Validation effectuée.
```

La réservation se déroule avec succès.

à ce niveau, le client 1 va réserver 2 places dans T1 mais puisque T2 a déjà écrit dans la table Client. La transaction T1 sera rejetée.

```
SQL> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
Session modifiée.
SQL> SELECT * FROM Vol;

IDVOL                                CAPACITEVOL  NBRPLACESRESERVEESVOL
-----
V1                                    150          0
SQL> UPDATE client
  2 SET nbrPlacesReserveesClient = nbrPlacesReserveesClient + 2
  3 WHERE idClient = 'C1';
UPDATE client
*
ERREUR à la ligne 1 :
ORA-08177: impossible de sérialiser l'accès pour cette transaction
```

Ce ne sera pas le même algorithme que le verrouillage à deux phases. Car s'il s'agissait du verrouillage à deux phases. La transaction T2 resterait en attente après la première lecture de T1. Dans notre cas, T2 a réalisé l'écriture($w_2(d), w_2(d')$) après la lecture de T1($r_1(d)$). Le verrouillage à deux phases n'aurait pas permis la réalisation de ce cas.