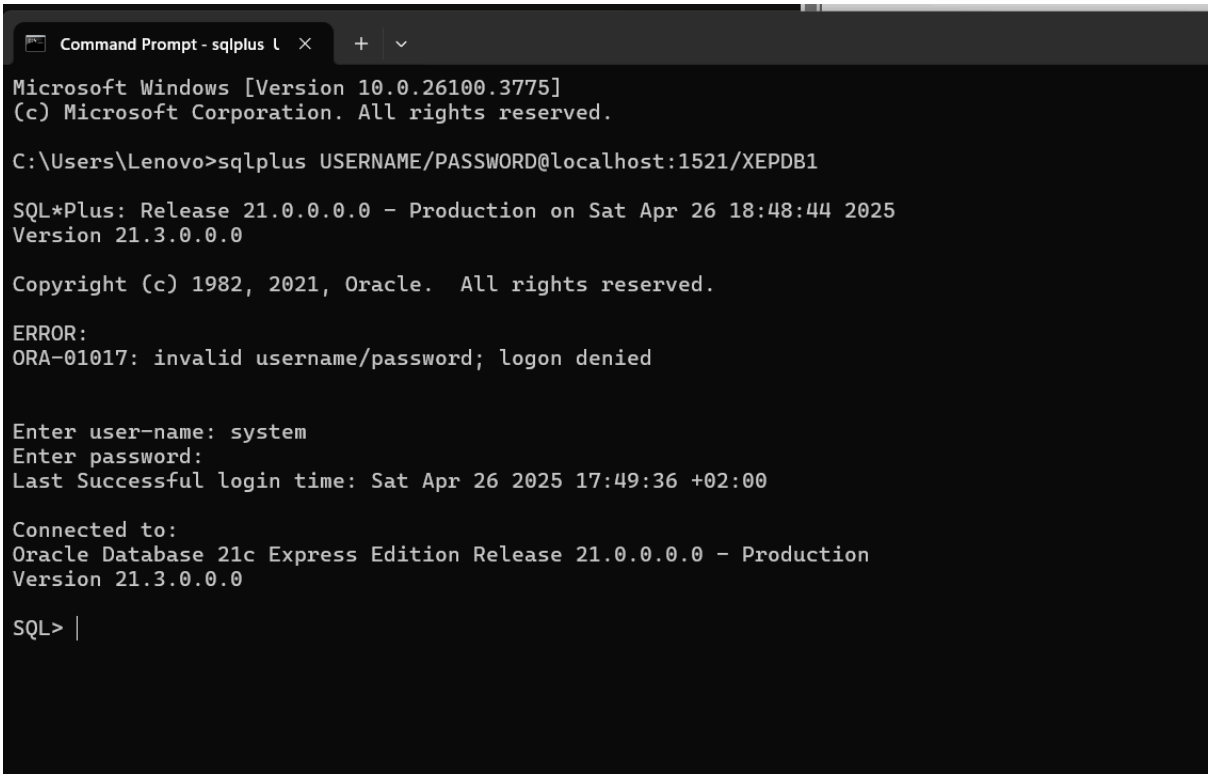
Yosra SASSI 2ING

Compte Rendu – TP n°5 : Transactions et Contrôle de Concurrence

Environnement de travail :

Connexion établie via SQL*Plus en mode commande :



Puis exécution de :

```
SQL> SET AUTOCOMMIT OFF;
```

pour désactiver la validation automatique des transactions.

Exercice 1 – Atomicité d'une transaction

1. Création de la table transaction

Commande exécutée en session S1 :

```
SQL> CREATE TABLE transaction(
  2       idTransaction VARCHAR2(44),
  3       valTransaction NUMBER(10)
  4  );

Table created.
```

2. Opérations sur la table dans la session S2

- Insertion de lignes :

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

1 row created.

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

1 row created.
```

- Modification d'une ligne :

```
SQL> UPDATE transaction SET valTransaction = 150 WHERE idTransaction = 'T1';

1 row updated.
```

- Suppression d'une ligne :

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

1 row deleted.
```

- ROLLBACK :

```
SQL> ROLLBACK;

Rollback complete.
```

➔ **Résultat :** toutes les modifications effectuées dans S2 sont annulées. La table reste vide.

3. Insertions puis fermeture avec QUIT sans COMMIT (Session S2)

Insertion de nouvelles lignes dans S2 :

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

1 row created.

SQL> INSERT INTO transaction VALUES ('T4', 400);

1 row created.
```

Commande :

```
SQL> QUIT;
Disconnected from Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0
```

➔ **Résultat en S1 :** En consultant la table depuis S1 (SELECT * FROM transaction;), aucune nouvelle ligne n'est visible. Les données ont été perdues car il n'y a pas eu de validation (COMMIT).

4. Insertions puis fermeture brutale (Session S1)

```
SQL> INSERT INTO transaction VALUES ('T5', 500);

1 row created.
```

➔ **Résultat :** À la reconnexion, les données ne sont pas présentes. Sans COMMIT, tout travail est perdu en cas de fermeture brutale.

5. Modification de structure puis ROLLBACK

```
SQL> ALTER TABLE transaction ADD (val2Transaction NUMBER(10));

Table altered.
```

Exécution d'un ROLLBACK :

```
SQL> ROLLBACK;

Rollback complete.
```

➔ **Résultat :** La modification de la structure de la table est conservée malgré le ROLLBACK.

Les commandes DDL (comme CREATE, ALTER) sont automatiquement validées dans Oracle.

Exercice 5 : Isolation complète et verrouillage

- Dans cet exercice, nous étudions l'impact de l'exécution concurrente de transactions sur la cohérence des données dans une base Oracle, en expérimentant différents niveaux d'isolation (READ COMMITTED par défaut et SERIALIZABLE).

- Nous utilisons deux sessions SQL*Plus (T1 et T2) pour réaliser des opérations simultanées sur deux tables : client et vol.

1. Travailler d'abord avec READ COMMITTED (par défaut)

- Deux sessions ouvertes : T1 et T2.

2. Simulation de deux transactions concurrentes

Préparation de l'environnement :

Dans chaque session ouverte (S1 et S2) :

```
SQL> SET AUTOCOMMIT OFF;
```

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

Table created.

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

Table created.
```

Séquence d'exécution :

- T1 (client C1 réserve 2 billets) :

```
SQL> SELECT * FROM vol WHERE idVol = 'V1';

IDVOL                                    CAPACITEVOL NBRPLACESRESERVEESVOL
---------------------------------------- ----------- ---------------------
V1                                               100                     0

SQL> SELECT * FROM client WHERE idClient = 'C1';

IDCLIENT                                 PRENOMCLIEN
---------------------------------------- -----------
NBRPLACESRESERVEESCLIENT
------------------------
C1                                       Alice
                       0
```

- T2 (client C2 réserve 3 billets) :

```
SQL> SELECT * FROM vol WHERE idVol = 'V1';

IDVOL                                    CAPACITEVOL NBRPLACESRESERVEESVOL
---------------------------------------- ----------- ---------------------
V1                                               100                     0

SQL> SELECT * FROM client WHERE idClient = 'C2';

IDCLIENT                                 PRENOMCLIEN
---------------------------------------- -----------
NBRPLACESRESERVEESCLIENT
------------------------
C2                                       Bob
                       0
```

- T1 :

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

1 row updated.

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

1 row updated.

SQL> COMMIT;

Commit complete.
```

- T2 :

```
SQL> UPDATE client SET nbrPlacesReserveesClient = nbrPlacesReserveesClient + 3 WHERE idClient = 'C2';

1 row updated.

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

1 row updated.

SQL> COMMIT;

Commit complete.
```

Résultat final attendu en READ COMMITTED :

- Le vol n'enregistre pas la somme 2 + 3 = 5 billets, mais seulement les derniers billets réservés. ➔ Incohérence dans les données.

Conclusion

Ce TP nous a permis de comprendre l'importance de l'atomicité des transactions, du contrôle de concurrence, et des niveaux d'isolation en SQL. Il a également mis en évidence les risques d'incohérence sous READ COMMITTED et la protection apportée par le mode SERIALIZABLE.