Tp4

ex1 :

1. Sous le super admin SYS, créer une table etudiants et y insérer quelques lignes, consulter le contenu de la table, modifier une ligne, supprimer une autre et enfin annuler les mises à jour venant d'être effectuées avec la commande ROLLBACK. Voir à nouveau le contenu de la table et sa structure. Pourquoi la table est-elle vide ?

```
-- Connexion sous SYS (en tant que SYSDBA)
CONNECT SYS AS SYSDBA;

-- Création de la table
CREATE TABLE etudiants (
    id NUMBER PRIMARY KEY,
    nom VARCHAR2(50)
);

-- Insertion de quelques lignes
INSERT INTO etudiants VALUES (1, 'Ali');
INSERT INTO etudiants VALUES (2, 'Sana');
INSERT INTO etudiants VALUES (3, 'Mourad');

-- Affichage
SELECT * FROM etudiants;

-- Modification
UPDATE etudiants SET nom = 'Sami' WHERE id = 1;

-- Suppression
DELETE FROM etudiants WHERE id = 2;

-- Annulation
ROLLBACK;

-- Vérification
SELECT * FROM etudiants;

-- Structure
DESC etudiants;
```

❓ Pourquoi la table est-elle vide ?

Parce que les `INSERT`, `UPDATE`, et `DELETE` ont tous été annulés par le `ROLLBACK` : ils faisaient .tous partie de la même transaction non validée

2. Insérer à nouveau quelques lignes dans Etudiants, les modifier et les détruire partiellement, puis valider ces mises à jour avec la commande COMMIT, puis déclencher un ROLLBACK. Que s'est-il passé ? Maintenant détruire les données de la table et valider.

```
-- Insertion
INSERT INTO etudiants VALUES (4, 'Hana');
INSERT INTO etudiants VALUES (5, 'Yassine');

-- Modification
UPDATE etudiants SET nom = 'Ines' WHERE id = 5;

-- Suppression
DELETE FROM etudiants WHERE id = 4;

-- Validation
COMMIT;

-- Tentative d'annulation
ROLLBACK;

-- Résultat
SELECT * FROM etudiants;
```

❓ Que s’est-il passé ?

Le `COMMIT` a validé définitivement toutes les modifications. Le `ROLLBACK` après un `COMMIT` .n’annule rien, car la transaction a été clôturée. Les données supprimées restent supprimées

3. Insérer à nouveau dans Etudiants quelques lignes et clore la transaction par un EXIT ou un QUIT. Que s’est-il passé ?

```
-- Fermeture
EXIT;
```

❓ Que s’est-il passé ?

Sans `COMMIT`, les données ne sont pas sauvegardées. Oracle effectue implicitement un `ROLLBACK` .à la fermeture si aucun COMMIT n’a été fait

4. Insérer à nouveau deux ou trois lignes dans la table Etudiants et fermer brutalement Sql Command Line, puis rentrer à nouveau dans votre compte. Les données saisies ont-elles été préservées ?

```
-- Insertion
INSERT INTO etudiants VALUES (7, 'Nadia');

-- Fermeture du terminal brutalement (ex : Ctrl+C ou fermeture fenêtre)
```

❓ Données préservées ?

Non. Comme aucun `COMMIT` n’a été fait, Oracle a annulé implicitement la transaction. Les données .sont perdues

5. Insérer à nouveau quelques lignes dans la table Etudiants, puis adjoindre une nouvelle colonne à sa table (ou plus généralement émettre n’importe quelle commande de description des données) et essayer d’annuler les dernières insertions. Pourquoi est-ce impossible ?

```
-- Insertion sans COMMIT
INSERT INTO etudiants VALUES (6, 'Omar');
-- Insertion
INSERT INTO etudiants VALUES (8, 'Walid');
```

-- Modification de la structure
ALTER TABLE etudiants ADD age NUMBER;

-- Tentative d'annulation
ROLLBACK;

-- Résultat
SELECT * FROM etudiants;

❓ Pourquoi l'insertion n’a pas été annulée ? Car les commandes DDL (comme `ALTER`, `CREATE`, `DROP`) déclenchent automatiquement un COMMIT avant et après leur exécution. Donc :

- Le `ALTER TABLE` a validé toutes les insertions précédentes
- `ROLLBACK` après cela ne peut rien annuler

Résultat	**ROLLACK possible ?**	**Avec COMMIT ?**	**Action**
Tout annulé	OUI	NON	Insertion/modif/suppression
Changements conservés	NON	OUI	Insertion/modif/suppression
Changements perdus	Implicite ROLLBACK	NON	Insertion sans COMMIT, puis EXIT
Changements conservés (auto-COMMIT)	NON	NON	Insertion, puis DDL (ex: ALTER TABLE)

Ex2 :

```
CREATE TABLE members (
  member_id NUMBER PRIMARY KEY,
  first_name VARCHAR2(50),
  last_name VARCHAR2(50)
);
```

2. Se connecter sous le schéma où se trouve la table members

-- Connexion à l’utilisateur normal (ex. USER1)

CONNECT user1/password;

3. Afficher toute la table members ainsi que le nombre des lignes

-- Affichage des données

SELECT * FROM members;

SELECT COUNT(*) FROM members;

4. Ouvrir une autre fenêtre de run sql command line et se connecter avec le super admin sys

CONNECT sys/password AS SYSDBA;

5. Afficher maintenant la table members

SELECT * FROM user1.members;

6. On retourne à la première fenêtre, insérer un nouveau tuple dans la table members

INSERT INTO members VALUES (10, 'Lina', 'Ben Amor');

7. Réafficher la table members depuis le super admin SYS (dans la deuxième fenêtre). Que remarquez-vous ?

SELECT * FROM user1.members;

📌 Observation : la ligne n'apparaît pas ! Car elle n'a pas encore été validée (`COMMIT`) dans la première session.

8. Faire le nécessaire pour valider la transaction (l'insertion) et voir si la modification est apparue depuis le super admin ou pas

COMMIT;

SELECT * FROM user1.members;

📌 Maintenant la ligne est visible dans toutes les sessions. `COMMIT` rend la transaction permanente et visible globalement.

9. Dans la première fenêtre, insérer de nouveau une autre ligne dans members

INSERT INTO members VALUES (11, 'Ahmed', 'Sassi');

SELECT COUNT(*) FROM members;

10. Afficher le nombre des lignes de la table

ROLLBACK;

SELECT COUNT(*) FROM members;:

11. Faisons un ROLLBACK maintenant. Que remarquez-vous ?

📌 Observation : la ligne insérée (Ahmed) a été annulée et n'est plus visible.

12. Essayer cette transaction dans la première fenêtre : SQL> delete from members; SQL> select count(*) from members;

DELETE FROM members;

SELECT COUNT(*) FROM members;

13. Maintenant dans la 2eme fenêtre, afficher le nombre des lignes de la table members. Que remarquez-vous ?

SELECT COUNT(*) FROM user1.members;

📌 Observation : la suppression n'est pas visible dans la 2e fenêtre (car non validée).

14. Annuler la transaction de suppression dans la première fenêtre

ROLLBACK;

SELECT COUNT(*) FROM members;

📌 Les lignes supprimées sont récupérées.

15. Insérer une nouvelle ligne dans la table members et se faire déconnecter d'une façon normale a l'aide de la commande exit.

INSERT INTO members VALUES (12, 'Yosra', 'Trabelsi');

EXIT;

SELECT * FROM user1.members;

📌 **Observation :** la ligne **n'est pas visible**, car aucun `COMMIT` n'a été fait → Oracle fait un **ROLLBACK implicite** à la déconnexion.

16. Dans la première fenêtre, supprimer toutes les lignes de la table members

DELETE FROM members;

-- Ne pas faire de COMMIT ni ROLLBACK

17. Fermer la fenêtre d'une façon anormale (cliquer sur la croix de la fermeture). Maintenant, se reconnecter de nouveau avec l'utilisateur sys, et réafficher la table members. Que remarquezvous ?

SELECT COUNT(*) FROM user1.members;

📌 Les données n'ont pas été supprimées. Fermeture brutale = ROLLBACK implicite.

18. Changer le first_name des members_ID 3 et 2 en 'manel' et 'mohamed'.

UPDATE members SET first_name = 'Manel' WHERE member_id = 3;

UPDATE members SET first_name = 'Mohamed' WHERE member_id = 2;

19. Créer maintenant un savepoint qui porte le nom ‘ updations’. N’importe quelle transaction exécutée avant ce savepoint, elle va être protégée par ce savepoint updations.

SAVEPOINT updations;

20. Insérer une nouvelle ligne et créer un savepoint INSERTIONS. Afficher la table

INSERT INTO members VALUES (13, 'Karim', 'Lahmar');

SAVEPOINT insertions;

SELECT * FROM members;

21. Faire un rollback vers insertions. Afficher la table. Que remarquez-vous ?

ROLLBACK TO insertions;

SELECT * FROM members;

📌 La ligne (Karim) est retirée ; on est revenu à l'état juste après `updations`.

22. Faire un rollback vers updations. Que remarquez-vous ?

ROLLBACK TO updations;

SELECT * FROM members;

📌 Reviens à l’état juste après la mise à jour de `first_name`. Toutes les insertions ultérieures sont annulées.

23. Valider la transaction de la mise à jour. Maintenant passons à la deuxième fenêtre du schéma sys. Afficher la table members

COMMIT;

SELECT * FROM user1.members;

📌 Toutes les mises à jour sont **désormais visibles** dans la session SYS.

Opération	Effet
`INSERT`, `UPDATE`	Pas visibles tant que pas validés
`COMMIT`	Rend les modifications visibles globalement
`ROLLBACK`	Annule toute la transaction active
`SAVEPOINT`	Marque un point dans la transaction
`ROLLBACK TO savepoint`	Annule jusqu’à un point précis seulement
Fermeture sans `COMMIT`	Annulation implicite (`ROLLBACK`)
DDL (`ALTER`, etc.)	Force un `COMMIT` avant et après

Exercice 3 :

1. Se connecter sous SYS

CONNECT sys/password AS SYSDBA;

2. Supprimer la table Etudiants (déjà crée lors de l'exercice 1), puis la recréer avec un NOM VARCHAR2(16) et ajouter quelques lignes. Ensuite, se connecter à son compte à partir d'une autre instance et consulter à travers cette nouvelle fenêtre le contenu du compte. Que voyez-vous ?

DROP TABLE etudiants;

CREATE TABLE etudiants (

  nom VARCHAR2(16)

);

INSERT INTO etudiants VALUES ('Ali');

INSERT INTO etudiants VALUES ('Sana');

INSERT INTO etudiants VALUES ('Hatem');

-- Ne pas COMMIT

⚠ **Ne pas exécuter `COMMIT`** pour tester la visibilité dans d'autres sessions.

3. Insérer une ligne dans la deuxième fenêtre. Afficher etudiants dans les deux tables. Que voiton?

CONNECT user1/password;

SELECT * FROM sys.etudiants;

📌 Observation :

Vous ne verrez probablement aucune donnée dans la table `etudiants` si la première •
session n'a pas encore fait de `COMMIT`, car Oracle applique le principe de lecture cohérente :
.chaque session voit les données validées au début de sa propre transaction

4. Créer dans l'une des deux fenêtres ouvertes une nouvelle table UE : (NomUE VARCHAR2(30) , Nbetudiants INTEGER) et y insérer par chacune des fenêtres quelques lignes. Que voit-on de la table Etudiants ?

INSERT INTO sys.etudiants VALUES ('Marwa');

SELECT * FROM sys.etudiants;

📌 Observation :

.Fenêtre 1 ne voit que ses propres insertions non validées•

.′Fenêtre 2 voit uniquement ‘Marwa•

.Isolation des transactions : chaque session a sa propre vision des données•

5. Détruire la table UE dans une des fenêtres. Que se passe-t-il ? Consulter le contenu de la table UE depuis l’autre fenêtre. Que contient-elle désormais ? Comment détruire UE ?

CREATE TABLE ue (

  nomUE VARCHAR2(30),

  nbetudiants INTEGER

);

INSERT INTO ue VALUES ('Maths', 50);

INSERT INTO ue VALUES ('Physique', 40);

📌 Observation :

.Tant que personne ne fait de `COMMIT`, les insertions sont isolées•

: Si vous affichez `SELECT * FROM ue;` dans chaque fenêtre•

′Fenêtre 1 verra uniquement ‘Maths•

′Fenêtre 2 verra uniquement ‘Physique•

6. Adjoindre une clé primaire à Etudiants. Insérer une même ligne dans la première fenêtre puis dans la deuxième. Que se passe-t-il ? Émettre un ROLLBACK dans la première fenêtre. Que devient le blocage ?

DROP TABLE ue;

SELECT * FROM ue;

📌 **Erreur probable** : *"table or view does not exist"*

Pourquoi ? Car Oracle applique un **verrou DDL exclusif** : une fois qu’une session a supprimé la table, **elle n’existe plus dans la base**, même si une autre session y avait encore une transaction .ouverte

Pour éviter cela : il faut que les **deux sessions fassent un COMMIT** avant suppression, ou 🛠️ .qu’on attende que les verrous soient levés

7 :

ALTER TABLE etudiants ADD CONSTRAINT pk_etudiants PRIMARY KEY (nom);

INSERT INTO etudiants VALUES ('Zied');

INSERT INTO etudiants VALUES ('Zied');

📌 **Résultat :** la 2e insertion **bloque ou échoue** à cause de la **contrainte d’unicité**. Si la 1re transaction n’est pas encore validée, la seconde est **bloquée** (en attente de libération du verrou).

8 :

ROLLBACK;

📌 **Effet :**

.Oracle **libère le verrou**•

.Fenêtre 2 peut maintenant **insérer ‘Zied’** avec succès•

Comportement observé	**Concept SQL Oracle**
Une session ne voit que les données **validées**	Lecture cohérente
Insertions/modifications visibles uniquement localement	Transactions isolées
Une table supprimée n’existe plus pour les autres sessions	Verrouillage DDL
Empêche les doublons dans plusieurs sessions	Clé primaire + blocage
Libère les ressources/verrous	ROLLBACK et déblocage