Jules GINHAC et Dorian TONNIS

Partie 1 : Transactions

Atomicité d'une transaction courante

1° Créer une table et manipuler des données :

- **Création de la table et insertion de lignes :**

```
CREATE TABLE MaTable (id INT PRIMARY KEY, nom VARCHAR(50));
INSERT INTO MaTable (id, nom) VALUES (1, 'Ligne 1');
INSERT INTO MaTable (id, nom) VALUES (2, 'Ligne 2');
INSERT INTO MaTable (id, nom) VALUES (3, 'Ligne 3');
SELECT * FROM MaTable;
```

- **Modification, suppression et annulation :**

```
UPDATE MaTable SET nom = 'Nouveau Nom' WHERE id = 1;
DELETE FROM MaTable WHERE id = 2;
SELECT * FROM MaTable;
ROLLBACK;
SELECT * FROM MaTable;
```

2° Valider et annuler des mises à jour :

- **Mises à jour, validation, puis ROLLBACK :**

```
INSERT INTO MaTable VALUES (4, 'Ligne 4'), (5, 'Ligne 5');
UPDATE MaTable SET nom = 'Modifié' WHERE id = 4;
DELETE FROM MaTable WHERE id = 5;
COMMIT;
ROLLBACK;
```

  La validation a empêché le ROLLBACK de supprimer les modifications.

- **Suppression des données :**

```
DELETE FROM MaTable;
COMMIT;
SELECT * FROM MaTable;
```

3° Fermer une transaction avec EXIT ou QUIT :

- **Insérer des lignes et clore avec EXIT/QUIT :**

```
INSERT INTO MaTable (id, nom) VALUES (1, 'Ligne 1');
INSERT INTO MaTable (id, nom) VALUES (2, 'Ligne 2');
INSERT INTO MaTable (id, nom) VALUES (3, 'Ligne 3');
EXIT;
ROLLBACK;
```

Nous avons été déconnectés de la base de données et les modifications ont été validées.

4° Fermer brutalement la session et revenir :

- **Fermer brutalement et vérifier la persistance des données :**

```
INSERT INTO MaTable (id, nom) VALUES (4, 'Ligne 4');
INSERT INTO MaTable (id, nom) VALUES (5, 'Ligne 5');
```

La session a été fermée brutalement et les modifications n'ont pas été validées.

5° Manipuler les données et la structure de la table :

- **Insertions, ajout de colonne et annulation :**

```
INSERT INTO MaTable (id, nom) VALUES (6, 'Ligne 6');
INSERT INTO MaTable (id, nom) VALUES (7, 'Ligne 7');
ALTER TABLE MaTable ADD nouvelle_colonne INT;
ROLLBACK;
```

La colonne a bien été ajoutée mais le ROLLBACK a supprimé toutes les données.

6° Conclusion sur les transactions :

- **Qu'est-ce qu'une transaction et comment valider ou annuler :**
  Une transaction est une unité de travail. Lorsqu'une transaction aboutit, toutes les modifications sont validées mais si une transaction rencontre des erreurs et doit être annulée ou restaurée,

toutes les modifications de données sont supprimées.
On peut valider les changements avec `COMMIT` et les annuler avec `ROLLBACK`.

Plusieurs sessions sur un seul compte de BD et transactions concurrentes

1° Connexion depuis une autre fenêtre :

- **Se connecter depuis une autre fenêtre :**
  Toutes les données validées sont visibles.

2° Insérer des lignes depuis les deux fenêtres :

- **Insérer des lignes depuis les deux fenêtres :**
  Les modifications faites dans l'autre fenêtre ne sont pas visibles. On ne peut les voir qu'après validation avec `COMMIT`.

3° Créer une nouvelle table depuis une fenêtre :

- **Créer une nouvelle table et insérer des lignes :**
  La nouvelle table est visible dans l'autre fenêtre mais les lignes insérées ne le sont pas.

4° Détruire la nouvelle table :

- **Détruire la nouvelle table :**
  La table est détruite dans les deux fenêtres.

5° Ajouter une clé à la table et tester le ROLLBACK :

- **Ajouter une clé et tester le ROLLBACK :**
  Sur la deuxième fenêtre à faire la modification, SQLDeveloppper se bloque. Tant que les modifications ne sont pas validées ou annulées sur la première fenêtre, la deuxième fenêtre reste blockée.

6° Fermer la session dans la fenêtre d'insertion :

- **Fermer la session avec EXIT ou QUIT :**
  Quand on se déconnecte proprement, les modifications sont validées et visibles dans les autres fenêtres.

7° Ouvrir une nouvelle session :

- **Ouvrir une nouvelle session et vérifier la dernière transaction :**
  Les modifications de la dernière transaction sont visibles dans la nouvelle session.

8° Insérer une ligne, créer une nouvelle table et ROLLBACK :

- **Insérer une ligne, créer une nouvelle table et ROLLBACK :**
  La création de la nouvelle table a validé les données. On peut donc voir la ligne insérée avant la création de la table et la nouvelle table mais pas la ligne insérée après la création de la table.

9° Insérer une ligne, éliminer une table et ROLLBACK :

- **Insérer une ligne, éliminer une table et ROLLBACK :**
  La suppression de la table a validé les données. On peut donc voir la ligne insérée avant la suppression de la table et la table a bien été supprimée.

Droits/privilèges entre deux comptes d'une même base de données

1° Accorder le droit SELECT à l'autre groupe :

- **Accorder le droit SELECT à l'autre groupe :**

```
-- Dans le groupe 02 :
CREATE TABLE AutreGroupe (id INT PRIMARY KEY, nom VARCHAR(50));
INSERT INTO AutreGroupe (id, nom) VALUES (1, 'Ligne 1');
INSERT INTO AutreGroupe (id, nom) VALUES (2, 'Ligne 2');
INSERT INTO AutreGroupe (id, nom) VALUES (3, 'Ligne 3');
GRANT SELECT ON AutreGroupe TO INI3A09;
```

Vérifier le privilège accordé :

```
-- Dans le groupe 09 :
SELECT * FROM ALL_TABLES;
SELECT * FROM INI3A02.AutreGroupe;
```

Le groupe 09 peut voir la table du groupe 02.

78	INI3A09	COULEUR	USERS	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2
79	INI3A02	AUTREGROUPE	USERS	(null)	(null)	VALID	10	(null)	1	255	(null)	(null)	(null)	
80	INI3A09	PC	USERS	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2

2° Observer les mises à jour de l'autre groupe :

- **Observer les mises à jour de l'autre groupe :**
  Si le groupe 02 insère une ligne dans la table et fait un `COMMIT`, le groupe 09 peut la voir mais pas la modifier.

3° Essayer d'insérer une ligne dans la table de l'autre groupe :

- **Essayer d'insérer une ligne sans le droit INSERT :**
  Le groupe 09 ne peut pas insérer de ligne dans la table du groupe 02.

```
Erreur commençant à la ligne: 76 de la commande -
insert into ini3a02.AutreGroupe values (4,'Ligne 4')
Erreur à la ligne de commande: 76 Colonne: 21
Rapport d'erreur -
Erreur SQL : ORA-01031: privilèges insuffisants
01031. 00000 -  "insufficient privileges"
*Cause:    An attempt was made to perform a database operation without
           the necessary privileges.
*Action:   Ask your database administrator or designated security
           administrator to grant you the necessary privileges
```

4° Accorder le droit INSERT par l'autre groupe :

- **Accorder le droit INSERT par l'autre groupe :**

```sql
-- Dans le groupe 02
GRANT INSERT ON AutreGroupe TO INI3A09;
```

- **Reprendre l'insertion :**

```sql
-- Dans le groupe 09
INSERT INTO INI3A09.AutreGroupe VALUES (4, 'Ligne 4');
```

Le groupe 09 peut maintenant insérer des lignes dans la table du groupe 02.
Résultat pour le groupe 02 :

	ID	NOM
1	4	Ligne 4
2	1	Ligne 1
3	2	Ligne 2
4	3	Ligne 3

5° Réaliser une jointure entre les tables des deux groupes :

- **Réaliser une jointure entre les tables des deux groupes :**

```
-- Dans le groupe 09
CREATE TABLE pc(pc VARCHAR(20), id INT, PRIMARY KEY (id));
INSERT INTO pc VALUES ('pc1' , 1);
INSERT INTO pc VALUES ('pc2' , 2);
INSERT INTO pc VALUES ('pc3' , 3);
SEELCT * FROM INI3A02.AutreGroupe a JOIN pc p ON p.id = a.id;
```

	ID	NOM	PC	ID_1
1	1	Ligne 1	pc1	1
2	2	Ligne 2	pc2	2
3	3	Ligne 3	pc3	3

Partie 2 : PLSQL

1. Copier les tables dans votre compte :

```
CREATE TABLE Dept AS SELECT * FROM Scott.Dept;
CREATE TABLE Emp AS SELECT * FROM Scott.Emp;
CREATE TABLE Salgrade AS SELECT * FROM Scott.Salgrade;

SELECT * FROM Dept;
SELECT * FROM Emp;
SELECT * FROM Salgrade;
```

	DEPTNO	DNAME	LOC	NBEMPS
1	10	ACCOUNTING	NEW YORK	3
2	20	RESEARCH	DALLAS	5
3	30	SALES	CHICAGO	6
4	40	OPERATIONS	BOSTON	0

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	17/12/80	800	(null)	20
2	7499	ALLEN	SALESMAN	7698	20/02/81	1600	300	30
3	7521	WARD	SALESMAN	7698	22/02/81	1250	500	30
4	7566	JONES	MANAGER	7839	02/04/81	2975	(null)	20
5	7654	MARTIN	SALESMAN	7698	28/09/81	1250	1400	30
6	7698	BLAKE	MANAGER	7839	01/05/81	2850	(null)	30
7	7782	CLARK	MANAGER	7839	09/06/81	2450	(null)	10
8	7788	SCOTT	ANALYST	7566	19/04/87	3000	(null)	20
9	7839	KING	PRESIDENT	(null)	17/11/81	5000	(null)	10
10	7844	TURNER	SALESMAN	7698	08/09/81	1500	0	30
11	7876	ADAMS	CLERK	7788	23/05/87	1100	(null)	20
12	7900	JAMES	CLERK	7698	03/12/81	950	(null)	30
13	7902	FORD	ANALYST	7566	03/12/81	3000	(null)	20
14	7934	MILLER	CLERK	7782	23/01/82	1300	(null)	10

	GRADE	LOSAL	HISAL
1	1	700	1200
2	2	1201	1400
3	3	1401	2000
4	4	2001	3000
5	5	3001	9999

2. Requêtes SQL :

a. Employés dirigés par 'King' :

```sql
SELECT EName FROM Emp WHERE Mgr = (SELECT EmpNo FROM Emp WHERE EName = 'KING');
```

	ENAME
1	JONES
2	BLAKE
3	CLARK

b. Employés dépendant de 'Jones' :

```
SELECT EName FROM Emp START WITH EName = 'JONES' CONNECT BY PRIOR EmpNo = Mgr;
```

	ENAME
1	JONES
2	SCOTT
3	ADAMS
4	FORD
5	SMITH

c. Employés dont dépend 'Jones' :

```
SELECT EName FROM Emp START WITH Mgr = (SELECT EmpNo FROM Emp WHERE EName = 'JONES') CONNECT BY
```

	ENAME
1	SCOTT
2	ADAMS
3	FORD
4	SMITH

d. Employés dépendant de 'Blake', sauf 'Blake' lui-même :

```
SELECT EName FROM Emp WHERE Mgr = (SELECT EmpNo FROM Emp WHERE EName = 'BLAKE') AND EmpNo != (SI
```

	ENAME
1	ALLEN
2	WARD
3	MARTIN
4	TURNER
5	JAMES

e. Employés dépendant de 'King' sauf ceux dépendant de 'Blake' :

```
SELECT EName FROM Emp
WHERE Mgr IN (SELECT EmpNo FROM Emp WHERE EName = 'KING')
AND Mgr NOT IN (SELECT EmpNo FROM Emp WHERE EName = 'BLAKE');
```

	ENAME
1	JONES
2	BLAKE
3	CLARK

3. Fonction PL/SQL pour le nombre d'employés par département :

```
CREATE OR REPLACE FUNCTION GetEmpCount(p_deptno Emp.DeptNo%TYPE) RETURN NUMBER
IS
  v_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_count FROM Emp WHERE DeptNo = p_deptno;
  RETURN v_count;
END;
/

-- Appeler la fonction dans un bloc PL/SQL
DECLARE
  v_result NUMBER;
BEGIN
  v_result := GetEmpCount(20); -- Exemple avec le département 20
  DBMS_OUTPUT.PUT_LINE('Nombre d''employés : ' || v_result);
END;
/
```

4. Ajouter la colonne NbEmps à la table Dept :

a. En utilisant la fonction stockée :

```
ALTER TABLE Dept ADD NbEmps NUMBER;

UPDATE Dept d
SET d.NbEmps = GetEmpCount(d.DeptNo);
```

b. En utilisant un curseur :

```
ALTER TABLE Dept ADD NbEmps NUMBER;

DECLARE
  CURSOR C1 IS SELECT DeptNo FROM Dept;
  v_count NUMBER;
BEGIN
  FOR C1_enr IN C1 LOOP
    v_count := GetEmpCount(C1_enr.DeptNo);
    UPDATE Dept SET NbEmps = v_count WHERE DeptNo = C1_enr.DeptNo;
  END LOOP;
END;
/
```

5. Déclencheur pour mettre à jour le nombre d'employés :

```
CREATE OR REPLACE TRIGGER UpdateEmpCount
AFTER INSERT OR DELETE OR UPDATE OF DeptNo ON Emp
FOR EACH ROW
BEGIN
  UPDATE Dept SET NbEmps = GetEmpCount(:NEW.DeptNo) WHERE DeptNo = :NEW.DeptNo;
END;
/
```

6. Procédure pour mettre à jour le département de toute une équipe :

```
CREATE OR REPLACE PROCEDURE UpdateTeamDept(p_empno Emp.EmpNo%TYPE, p_new_deptno Dept.DeptNo%TYPE
IS
BEGIN
  -- Mettre à jour le département de toute l'équipe
  UPDATE Emp SET DeptNo = p_new_deptno
  WHERE EmpNo IN (SELECT EmpNo FROM Emp START WITH EmpNo = p_empno CONNECT BY PRIOR EmpNo = Mgr)
END;
/
```