NAAJI Dorian & ARMANET Nathan – 3A INFO GROUPE 2

POLYTECH LYON

Dorian.naaji@etu.univ-lyon1.fr

NATHAN.ARMANET@ETU.UNIV-LYON1.fr

Rapport TP03

BD1

# Partie 1 : Transactions

## Atomicité d’une transaction courante

### Question 1

« ROLLBACK » a supprimer tous les changements effectués à la table (insertion, modification et suppression de données) mais pas la table elle-même.

### Question 2

« COMMIT » a enregistré les modifications apportées à la table, ce qui à empêcher « ROLLBACK » de les supprimer.

### Question 3

« EXIT » ou « QUIT » ferme simplement la connexion à la base de données sans apporter de modification à celle-ci.

### Question 4

La fermeture brutale de la session est comme un « ROLLBACK » ; elle supprimer toutes les modifications de données non enregistrées.

### Question 5

Cette fois ci, le « ROLLBACK » ne supprime rien. Cela est dû au fait que l’on a modifier la table en elle-même (ajout, modification ou suppression de colonnes ou contraintes). Ces modifications enregistrent les données tel qu’elle sont avant de modifier la table.

### Question 6

Une transaction courante est un changement dans les données d'une table. Elles sont validées par « COMMIT » et annulé par « ROLLBACK ». Toutes modification de la table entraine un « COMMIT ».

## Plusieurs sessions sur un seul compte de BD et transactions concurrentes

### Question 1

On constate que le contenu de la base est le même quel que soit la fenêtre utilisée

### Question 2

On voit juste les transactions effectuées à partir de la même fenêtre

### Question 3

On voit les modifications de la table initiale apporté par l'utilisateur qui a créé la nouvelle table.

On peut voir la nouvelle table sur les deux fenêtres.

On voit seulement les transactions effectuer à partir de notre fenêtre

### Question 4

La suppression de la table fonctionne.

### Question 5

Seulement le premier utilisateur essayant d’insérer les données y parvient. Le deuxième utilisateur effectuant une insertion rencontre en effet un blocage « infini ». Une fois que le premier utilisateur effectue un ROLLBACK, l’insertion du deuxième utilisateur est instantanément effectuée.

### Question 6

On constate que la commande « EXIT » a commit les modifications.

### Question 7

En sortant normalement, la dernière transaction a en effet été validée.

### Question 8

On insère une ligne puis on crée une table ; un COMMIT est donc effectué et la ligne insérée dans la première table ne peut plus être ROLLBACK.

On insère ensuite une ligne dans la nouvelle table que l’on vient de créer, puis on ROLLBACK. La transaction ajoutant la ligne à la nouvelle table est ainsi annulée, mais la ligne ajoutée à la première table est toujours présente.

### Question 9

La dernière ligne insérée est toujours présente. On en déduit que la commande « DROP TABLE » effectue un commit.

## Droits/privilèges entre deux compte d’une même base de données

### Question 1

On se donne les privilèges et on peut vérifier qu’ils ont bien été données grâce aux requêtes

*« SELECT \* FROM ALL\_OBJECTS WHERE OWNER = 'INI3A06'; »*

*« SELECT \* FROM ALL\_TABLES WHERE OWNER = 'INI3A06'; »*

*« SELECT \* FROM USER\_TAB\_PRIVS WHERE OWNER = 'INI3A06'; »*

*« select \* from INI3A06.TRANSACTION »*

### Question 2

Quand l’autre groupe fait une mise à jour sur sa table puis effectue un COMMIT, on peut voir les modifications.

### Question 3

On ne parvient pas à insérer une mise à jour sur la table de l’autre groupe car nous n’avons pas le droit d’insérer de données dans cette dernière.

### Question 4

Avec le droit d’insertion, notre commande fonctionne désormais.

### Question 5

On peut réaliser la requête suivante pour la jointure :

*« SELECT \* FROM INI3A06.TRANSACTION*

*INNER JOIN TESTQ5\_TP3 ON TESTQ5\_TP3.id = INI3A06.TRANSACTION.ID\_TRANSACTION; »*

# Partie 2 : PLSQL

## Copier dans votre compte les trois table Dept, Emp et Salgrade de Scott.

On effectue les requêtes suivantes :

*« CREATE TABLE EMP AS SELECT \* FROM SCOTT.emp; »*

*« CREATE TABLE DEPT AS SELECT \* FROM SCOTT.Dept; »*

*« CREATE TABLE SalGrade AS SELECT \* FROM SCOTT.SalGrade; »*

## Donner les requêtes SQL pour les questions suivantes :

### Donner le nom des employés dirigés directement par ‘King’.

« select E1.\* from EMP E1

join EMP E2 on E2.DEPTNO = E1.DEPTNO

where E2.ENAME = 'KING'

and E1.ENAME != 'KING'; »

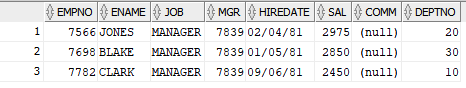


Figure : Résultat de la requête

### Donner le nom des employés qui dépendent (directement ou non) de Jones.

### Donner le nom des employés dont dépend (directement ou non) Jones.

### Donner le nom des employés dépendant de Blake, sauf Blake lui-même.

### Donner le nom des employés qui dépendent de King, sauf ceux qui dépendent de Blake.

## Ecrire une fonction PLSQL de paramètre un numéro de département et qui retourne le nombre d’employés de ce département. On pourra typer le numéro de département par *Emp.DeptNo%TYPE*.

## Ajouter la colonne NbEmps dans votre table Dept, qui contiendra le nombre d’employés de chaque département. On mettra à jour NbEmps de deux façons possibles:

### En utilisant la fonction stockée définie dans la question précédente

### Sans l’utiliser mais en utilisant un curseur associé à Dept suivant l’exemple suivant :

## Créer un déclencheur qui pour chaque ajout ou suppression d’un employé ou mise à jour du département d’un employé, modifiera le nombre d’employés dans la table Dept.