

TP Révisions

T. Libourel & I. Mougnot

1. Préambule

Oracle est accessible sur le serveur *Neptune* et vous êtes autorisés à accéder à une instance *master*. Vous utiliserez l'interface d'accès SQL*PLUS qui permet l'exécution de commandes spécifiques et de commandes SQL.

Remarque : Quelques commandes SQL*PLUS :

- SET pause ON/OFF défilement page par page
- SET LINESIZE n fixe le nbre de caractères de la ligne
- SET PAGESIZE n fixe le nbre de lignes par page
- DESC nomtable description de la table

Dictionnaire de données Nous rappelons que le dictionnaire de données est un ensemble de tables dans lesquelles sont stockées les descriptions des objets de la base. Les tables de ce dictionnaire peuvent être consultées au moyen du langage SQL. Des vues de ces tables permettent à l'utilisateur de voir les objets qui lui appartiennent (tables préfixées par USER) ou sur lesquels il a des droits (tables préfixées par ALL). L'administrateur a pour sa part accès à toutes les vues (les tables précédentes ainsi que les tables préfixées par DBA).

Quelques vues et tables du dictionnaire de données :

- USER_TABLES : tables et vues créées par l'utilisateur ;
- USER_CATALOG (ou CAT) : tables et vues sur lesquelles l'utilisateur a des droits à l'exception des tables et vues du dictionnaire de données ;
- USER_TAB_COLUMNS (ou COLS) : colonne de chaque table ou vue créée par l'utilisateur courant ;
- USER_CONSTRAINTS : définition des contraintes pour les tables des utilisateurs ;
- USER_CONS_COLUMNS : colonnes qui interviennent dans les définitions des contraintes ;
- USER_TAB_PRIVS : droits attribués et/ou reçus par l'utilisateur
- USER_SYS_PRIVS : privilèges donnés à l'utilisateur de manière générale ;
- USER_TAB_PRIVS_MADE : droits attribués par l'utilisateur ;
- USER_TAB_PRIVS_RECD : droits reçus par l'utilisateur ;
- ALL_CATALOG : liste de tous les objets accessibles par l'utilisateur ;
- ALL_TABLES Description des tables accessibles par l'utilisateur.

2. Création schéma et contraintes

On souhaite créer une base comportant les schémas relationnels (d'après R. Godin) :

- Client(noClient,nomClient,noTel)
- Article(noArticle, description,prixUnitaire,qtteEnStock)
- Commande(noCommande, dateCommande,noClient)
- LigneCommande(noCommande,noArticle,qtte)
- Livraison(noLivraison,dateLivraison)
- DetailLivraison(noLivraison,noCommande,noArticle,qtteLivr)

Le script de création sommaire est disponible.

Les contraintes suivantes ont été définies :

- Client : clé primaire noClient,
- Article : clé primaire noArticle ; contrôle qtteEnStock par défaut 0,
- Commande : clé primaire noCommande ; intégrité référentielle sur noClient de Client
- LigneCommande : clé primaire noCommande, noArticle ; intégrités référentielles : noCommande de Commande, noArticle d'Article ; contrôle qtte > 0
- Livraison : clé primaire noLivraison,
- DetailLivraison : clé primaire noLivraison, noCommande, noArticle ; intégrités référentielles noLivraison de Livraison, noCommande, noArticle de LigneCommande ; contrôle qtteLivr > 0

Connectez vous et effectuer les points suivants :

1. Créer les différentes contraintes sur le schéma précédent,
2. Tester la validité de ces contraintes en réalisant diverses insertions,
3. Donner toutes les informations sur les tables sur lesquelles vous avez des droits,
4. Donner le nombre d'attributs de la table Commande,
5. Donner la liste des contraintes (avec leur statut) créées
6. Donner les informations sur les contraintes de type clé primaire que vous avez créées

3. Gestion des accès concurrents

Une transaction (ensemble d'ordres SQL) est atomique c'est-à-dire qu'elle ne peut se terminer que par un succès (elle est alors validée COMMIT) ou un échec (tous ses effets sont alors détruits ROLLBACK).

En SQL une transaction se déroule selon les instructions

```
START TRANSACTION <mode>
<instruction lecture/écriture>
<instruction lecture/écriture>
<instruction lecture/écriture>
...
COMMIT; --- validation de toutes les écritures
START TRANSACTION <mode>
<instruction lecture/écriture>
<instruction lecture/écriture>
<instruction lecture/écriture>
...
ROLLBACK; --- annulation de toutes les écritures
```

Le mode peut être

- READ ONLY pour les transactions n'effectuant que des lectures
- READ WRITE pour les transactions effectuant lectures et écriture (mode par défaut d'ORACLE et pas d'instruction de début START)
- pouvant être complétés par l'introduction d'un niveau d'isolation grâce à l'instruction SET TRANSACTION ISOLATION LEVEL {niveau}, niveau pouvant être d'après SQL-92 READ UNCOMMITTED, READ COMMITED, REPETEABLE READ et SERIALIZABLE (en Oracle seuls READ COMMITED, et SERIALIZABLE)

En conséquence, en contexte multi-utilisateurs, les modifications effectuées par une transaction réalisée par un utilisateur ne sont connues des autres utilisateurs que lorsque la transaction a été confirmée par un COMMIT.

La concurrence d'accès entre transactions peut entraîner des phénomènes indésirables

- une transaction lit une donnée modifiée par une transaction concurrente non validée (et qui pourra donc l'annuler) : lectures "sales"
- une transaction peut lire deux fois une même donnée et celle-ci a changé car modifiée par une transaction concurrente : lecture non répétable
- une transaction peut effectuer une requête plusieurs fois avec obtention de résultats différents (données écrites par des transactions concurrentes) : lecture de fantômes

Les SGBD dont Oracle permettent de gérer la sérialisabilité de transactions concurrentes c'est-à-dire la synchronisation des exécutions concurrentes de façon à ce que leurs effets soient les mêmes que ceux d'une exécution en série grâce à un mécanisme de verrouillage.

Pour rester "simple" nous dirons que toute transaction pose des verrous sur les objets qu'elle manipule et que deux grands types de verrous existent :

- en lecture (verrou passant plusieurs lectures simultanées peuvent avoir lieu)
- en écriture (verrou bloquant la première écriture bloque les autres jusqu'à ce que le verrou soit relâché)

Commandes qui provoquent un blocage implicite sur les tables et les lignes impliquées : DELETE, INSERT, UPDATE, ALTER TABLE, ...

Remarque : en mode READ WRITE le niveau d'isolation READ COMMITED peut entraîner lectures non répétables et lecture de fantôme, le niveau d'isolation SERIALIZABLE interdit tout phénomène indésirable.

3.1 Privilèges d'accès à la base de données

Oracle permet à plusieurs utilisateurs de travailler sur la même base de données en toute sécurité. Deux commandes sont à ce titre particulièrement importantes : GRANT et REVOKE et permettent de définir les droits de chaque utilisateur sur les

objets de la base.

Tout utilisateur accède à la base à l'aide de son nom utilisateur et de son mot de passe. C'est le nom utilisateur qui permet de déterminer les droits d'accès aux objets de la base de données.

Les utilisateurs ont été créés par le DBA et sont autorisés à se connecter à la base Oracle MASTER. Ils ont aussi les privilèges de créer des objets de schéma de base de données utilisateur (tables, vues, contraintes etc).

Lorsque dans une session de TP vous avez travaillé sous le nom d'un utilisateur, vous n'avez donc pas été "en concurrence" avec d'autres utilisateurs.

Nous allons vérifier que le SGBD gère la concurrence d'accès à des objets de la base entre plusieurs utilisateurs (identiques ou différents).

Tout utilisateur qui crée des objets est propriétaire de ces objets (`table_name`, `owner` de `USER_tables` dans le dictionnaire des données). Le créateur d'un objet peut décider de donner (ou de supprimer) certains droits d'accès à cet objet à tout autre utilisateur de sa connaissance.

L'ordre GRANT GRANT privilege ON table/vue TO utilisateur [WITH GRANT OPTION]

Cet ordre permet de "donner" le privilège concerné sur la table ou la vue à l'utilisateur.

Exemple : X a créé la table TEST et veut autoriser Y à lire cette table.

Il passe alors l'ordre **GRANT SELECT ON TEST TO Y ;**

Les privilèges qui peuvent être donnés sont les suivants :

1. SELECT : droit de lecture ;
2. INSERT : droit d'insertion de lignes ;
3. UPDATE : droit de modification de lignes ;
4. DELETE : droit de suppression de lignes ;
5. ALTER : droit de modification de la définition de la table ;
6. INDEX : droit de création d'index ;
7. ALL : tous les droits ci-dessus.

Un utilisateur ayant reçu un privilège avec la mention facultative WITH GRANT OPTION peut les transmettre à son tour à un autre utilisateur.

L'ordre REVOKE Un utilisateur ayant accordé un privilège peut le reprendre à tout moment à l'aide de l'ordre REVOKE.

REVOKE privilege ON table/vue FROM utilisateur

Faire quelques essais. Par exemple, créer une table TEST(A smallint , B smallint, C varchar2(20)) et la contrainte TESTPK (clé primaire sur a) et insérer une dizaine de n-uplets dans TEST

- Donner (entre paire d'utilisateurs) des privilèges, vérifier l'obtention des privilèges, les tester ..
- Enlever les privilèges précédemment accordés
- Vérifier que les privilèges ont bien été supprimés

4. Tests sur transactions et concurrence

4.1 Transactions au sein d'une connexion unique

Une instruction du langage d'interrogation s'exécute en tout ou rien.

Voir l'effet du commit et du rollback Ouvrir une session SQLPLUS et tester (noter les observations) :

```
create table X (n Number (2)) ;
insert into X values (1) ;
select * from X ;
commit ;
select * from X ;
insert into X values (2) ;
select * from X ;
```

```
rollback ;
select * from X ;
commit ;
```

Effet des instructions du langage de définition et niveau d'isolation Un ordre du langage de définition des données forme une transaction à lui tout seul. L'exécution d'un tel ordre effectue un commit puis s'exécute et si tout s'est bien passé un nouveau commit est effectué, sinon (en cas d'erreur) la transaction n'est pas terminée. Tester et noter vos observations :

```
select * from X ;
insert into X values (55) ;
select * from X ;
create table Y (n Number (2)) ;
select * from X ;
insert into X values (56) ;
select * from X ;
rollback ;
select * from X ;
// constatation
insert into X values (57) ;
select * from X ;
create table Y (n Number (2)) ;
select * from X ;
// constatation
create table Y (n Number (2)) ;
set transaction isolation
  level read committed ;
// constatation
rollback ;
set transaction isolation
  level read committed ;
drop table X ;
drop table Y ;
// constatation
```

4.2 Transactions concurrentes pour un même utilisateur

Ouvrir deux connexions SQLPLUS. Vous êtes en concurrence (mais avec vous-même). Suivre les exemples proposés dans le tableau TAB 1.

Les modifications d'une transaction T sont invisibles des transactions **read committed** tant que T n'a pas validé (par commit).

D'autres exemples pourront être traités pour constater qu'un n-uplet peut "posséder" plusieurs versions ... et voir la différence entre read committed et serializable.

4.3 Concurrence entre utilisateurs différents

Pour la suite du TP vous allez donc fonctionner par "paire" (X connecté sur la machine m1 et Y connecté sur la machine m2).

- X (Y) donne les droits de lecture de "sa" table TEST à l'utilisateur Y (X)
- X (Y) donne les droits de modification de "sa" table à l'utilisateur Y (X)
- Vérifier que les privilèges ont été bien accordés
- Testez vos nouveaux droits (les objets que vous interrogerez et dont vous n'êtes pas propriétaire sont désignés par leur nom complet nompropriétaire.nomobjet).

Nous allons tester sur quelques exemples .

- tester les transactions concurrentes (selon le schéma du tableau 2) en mode isolation READ COMMITTED Que constatez-vous ?
- tester les transactions concurrentes (selon le schéma du tableau 3) en mode isolation READ COMMITTED Que constatez-vous ?
- tester les transactions concurrentes en mode READ ONLY. Que constatez-vous ? Mettez en concurrence une transaction READ ONLY et une transaction READ WRITE en niveau READ COMMITTED. Que constatez-vous ?

TAB. 1 – Utilisateur unique en concurrence

Connexion 1	Connexion 2
<pre> create table X (N Varchar2 (20)); insert into X values ('1 valeur init'); insert into X values ('2 valeur init'); select * from X; commit; insert into X values ('3 valeur init'); select * from X; commit; drop table X; </pre>	<pre> set transaction isolation level read committed; select * from X; select * from X; commit; <i>Conclusion ?</i> set transaction isolation level serializable; select * from X; select * from X; // constatation; commit; set transaction isolation level serializable; select * from X; // constatation; commit; </pre>

TAB. 2 – Deux utilisateurs différents -premier essai

T1	T2
<pre> UPDATE TEST set B= B-100 where A=2; UPDATE TEST set C= 'abc' where A=1; COMMIT; T3 SELECT * FROM TEST; SELECT * FROM TEST; COMMIT; </pre>	<pre> UPDATE TEST set B= B+40 where A=2; // blocage // déblocage UPDATE TEST set C= 'def' where A=1; COMMIT; </pre>

TAB. 3 – Deux utilisateurs différents - deuxième essai

T1	T2
<pre> UPDATE TEST set B= B-100 where A=2; UPDATE TEST set B= 0 where A=1; // blocage </pre>	<pre> UPDATE TEST set B= B-40 where A=1; UPDATE TEST set B= B-100 where A=2; //blocage </pre>
-----interblocage détecté-----	
si T1 avortée ROLLBACK;	// si T2 avortée ROLLBACK COMMIT;

- tester des transactions READ WRITE (selon le schéma du tableau 4) en niveau d'isolation SERIALIZABLE

TAB. 4 - Deux utilisateurs différents - troisième essai

T1	T2
<pre>SELECT A FROM TEST WHERE B=1; UPDATE TEST set B= B-100 where A=2; COMMIT;</pre>	<pre>SELECT A FROM TEST WHERE B=1; UPDATE TEST set B= B-100 where A=2; //rejet COMMIT;</pre>
<pre>T3 INSERT INTO TEST VALUES(20,100, 'aze'); UPDATE TEST SET B=20 WHERE A=25; COMMIT;</pre>	<pre>T4 INSERT INTO TEST VALUES(25,10, 'fgh'); UPDATE TEST SET B=10 WHERE A=20; //impossible</pre>