LIFBDW2 – Bases de données avancées
TP3 – Transactions et vues

Licence informatique – Automne 2016–2017

Les questions marquées du symbole (†) sont à préparer pour la séance

Exercice 1 : atomicité d'une transaction courante

1. Créer une table `Etudiants` et y insérer quelques lignes, consulter le contenu de la table, modifier une ligne, en 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 ?
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.
3. Insérer à nouveau dans `Etudiants` quelques lignes et clore la transaction par un `EXIT` ou un `QUIT`. Que s'est-il passé ?
4. Insérer à nouveau deux ou trois lignes dans la table `Etudiants` dans et fermer brutalement `SqlDeveloper`, puis rentrer à nouveau dans votre compte. Les données saisies ont-elles été préservées ?
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 ? Supprimer la colonne créée.
6. Insérer à nouveau quelques lignes dans la table `Etudiants`, puis fermer normalement `SqlDeveloper`. Une fenêtre apparaît, quels sont les choix proposés ?
7. En conclusion, qu'est-ce qu'une transaction courante et comment valider ou annuler une transaction ?

Exercice 2 : transactions concurrentes

1. Vider la table `Etudiants`, faire un `COMMIT` puis ajouter quelques lignes. Ensuite, se connecter à son compte à partir *d'une autre instance* de `SqlDeveloper` (ou sur un autre poste de travail) et consulter à travers cette nouvelle fenêtre le contenu du compte. Que voyez-vous ?
2. Insérer dans chacune des deux fenêtres deux ou trois lignes distinctes. Que voit-on de l'autre fenêtre ?
3. Créer dans l'une des deux fenêtres ouvertes une nouvelle table `UE` et y insérer par chacune des fenêtres quelques lignes. Que voit-on de la table `Etudiants` ?
4. 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` ?
5. 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 ?
6. Même question que précédemment : on insère depuis deux session différentes un même tuple qui violerait la contrainte de clé primaire, par contre, on émet un `COMMIT` dans la première. Que devient la seconde insertion ?
7. Ouvrir un nouveau fichier dans une instance de `SqlDeveloper` en utilisant la connexion déjà existante. Effectuer une insertion dans `Etudiants` dans un des onglets et consulter la table dans l'autre. Que voyez-vous ?
8. En conclusion, comment sont liés comptes, sessions et transactions ?

Exercice 3 : privilèges entre deux comptes

Les groupes de TP travaillent dans cette partie deux par deux.

1. Chaque groupe donne le droit à l'autre groupe de consulter l'une de ses tables en émettant un `GRANT SELECT ON Table TO autreGroupe ;`. Vérifier que ce privilège a été donné en effectuant une requête sur `ALL_TABLES` et `USER_TAB_PRIVS`. Consulter la table à laquelle l'autre groupe vous a donné accès.
2. Quand l'autre groupe fait une mise à jour sur sa table, que voyez-vous ?
3. Essayer d'insérer une ligne dans la table de l'autre groupe. Quel est le problème ?
4. Réaliser une requête utilisant d'une de vos tables avec une table de vos camarades.

Exercice 4 : création de vues

On considère le script de création de table suivant :

```
  CREATE TABLE TA
( ID_A NUMBER PRIMARY KEY,
  NAME_A VARCHAR2(32)
);

INSERT INTO TA VALUES(0,'foo');
INSERT INTO TA VALUES(1,'bar');

CREATE TABLE TB
( ID_B NUMBER PRIMARY KEY,
  NAME_B VARCHAR2(32),
  REF_A NUMBER REFERENCES TA(ID_A)
);

INSERT INTO TB VALUES(0,'foofoo',0);
INSERT INTO TB VALUES(1,'foobar',1);

COMMIT;
```

1. Créer une vue `TB _REF0` qui sélectionne tous les tuples de `TB` qui font référence au tuple $(0, \texttt{foo})$ de `TA`. Ajouter les tuples $(2, \texttt{foofoo2}, 0)$ $(3, \texttt{foobar2}, 1)$ à `TB _REF0` avec un `INSERT`. Quels sont les tuples ajoutés à `TB _REF0` et ceux ajoutés à `TB` ? Que constatez-vous ? Annulez les modifications pour revenir à l'état initial.
2. Créer maintenant une vue `TB_TA` qui fait la jointure de `TA` et `TB` sur les attributs `TB.REF_A` et `TA.ID_A`. Essayer d'insérer le tuple $(2, \text{foofoo2}, 0, 0, \text{foo})$. Quel est le problème ?
3. Essayer d'insérer dans `TB_TA` le tuple $(2, \text{foofoo2}, 0)$ qui n'est définit que pour les attributs des `TB`. Consulter la vue `TB _TA` et la table `TB`, que constatez-vous ? Essayer ensuite d'insérer ensuite le tuple $(3, \text{foofoo2}, 3)$, quel est le problème ? Annulez les modifications pour revenir à l'état initial.
4. Essayer d'insérer dans `TB_TA` le tuple $(2, \text{gaz})$ qui n'est définit que pour les attributs des `TA`. Quel est le problème ?
5. Reprendre la question précédente, mais cette fois sur une vue `TB_TA_ID` qui fait la jointure de `TB` et `TA` sur les attributs `TB.ID_B` et `TA.ID_A`. Pouvez-vous critiquer de la restriction imposée par Oracle sur ce type de mise-à-jour des vues ?
6. Créer une vue `TB_TA_2` similaire à celle de la question 1 avec les spécifications supplémentaires suivantes :
   — un attribut `NB_B` va compter pour chaque tuple de `TA` le nombre de tuples de `TB` qui lui font référence ;
   — les tuples de `TA` qui n'ont aucun tuple de `TB` correspondant doivent apparaître avec la valeur 0 pour `NB_B` et les attributs de `TB` à `NULL`