TP n° 5 : Transactions et contrôle de concurrence

Remarque En cas de difficulté à vous connectez rapportes-vous à la section préparation de l'environnement de travail de la série n° 1.

Pour ce TP, je vous conseille d'utiliser l'interface en ligne de commande *sqlplus* pour vous connecter à vos comptes.

Vous déposez le travail réalisé sur l'ENT.

Exercice n° 1 Atomicité d'une transaction

Commencer par ouvrir deux sessions (S_1 et S_2) et exécuter la commande suivants dans chacune des sessions (dans la suite de cet exercice, et à chaque connexion (ouverture de session), commencer par exécuter cette commande) :

```
SET AUTOCOMMIT OFF
```

1. Créer la table *transaction* dont le script est donné ci-dessous, en utilisant la session S_1.

```
CREATE TABLE transaction(
idTransaction VARCHAR2(44),
valTransaction NUMBER(10));
```

2. En utilisant la session S_2, insérer quelques lignes, modifier une ligne, en supprimer une autre et enfin annuler les mises à jour venant d'être effectuées avec la commande *ROLLBAK*. Afficher le contenu de la table.

3. Insérer à nouveau dans *transaction* quelques lignes et clore avec la commande *quit;*. Consulter le contenu de la table, en utilisant la session S_1. Que s'est-il passé ?

4. Dans la session S_1, insérer quelque lignes et fermer *brutalement sqlplus*. Reconnecter à nouveau sur votre compte. Les données saisies ont-elles été préservées ?

5. Dnas une nouvelle session, insérer quelques lignes et modifier la structure de la table *transaction*, en y ajoutant par exemple l'attribut *val2transaction* de type NUMBER(10). Exécuter la commande *ROLLBACK*. Que s'est-il passé ?

6. Conclure : définir de manière précise qu'est-ce qu'une session, une transaction et comment valider une transaction ou l'annuler.

Exercice n° 2 Transactions concurrentes

Créer le schéma de la base de données de gestion des billets d'avions, constitué des deux relations suivantes[1] : Client(idClient, prenomClient, placesReserveesClient) et
Vol (idVol, capaciteVol, placesReserveesVol)

```
CREATE TABLE vol(
idVol VARCHAR2(44),
capaciteVol NUMBER(10),
nbrPlacesReserveesVol NUMBER(10)
);
```

```
CREATE TABLE client(
idClient VARCHAR2(44),
prenomClient VARCHAR2(11),
nbrPlacesReserveesCleint NUMBER(10)
);
```

[1] Il s'agit d'un schéma simplifié mais suffisant, pour étudier les mécanismes transactionnels

Exécuter deux transactions T_1 et T_2 en parallèle (donc dans deux sessions différentes). Contrairement à l'exercice précédent où on a systématiquement utilisé *SET AUTOCOMMIT OFF*, avant le début de chaque session, dans cette exercice, on laisse le réglage d'isolation par défaut. Si vous utilisez ORACLE, ce réglage est *READ COMMITTED*. Et que certaines versions ne permettent pas d'obtenir les niveaux d'isolation *REPEATABLE READ* et *UNCOMMITTED READ*.

Insérer un vol et deux clients.

```
CREATE TABLE client(
idClient VARCHAR2(44),
prenomClient VARCHAR2(11),
nbrPlacesReserveesCleint NUMBER(10)
);
```

Isolation des transactions

Effectuer une réservation pour un client et ne validez pas encore cette transaction (T_1). Vous devriez constater que :

- les mises à jour sont visibles par la transaction qui les a effectuées (T_1)[2],
- elles sont en revanche invisibles par toute autre transaction (ici T_2).

C'est dû à la notion d'isolation des transactions. L'isolation est complète quand le résultat d'une transaction ne dépend pas de la présence ou non d'autres transactions.

COMMIT et ROLLBACK

Effectuez un *ROLLBACK* de la transaction en cours (T_1). Vous devriez constater :

- que la base revient à son état initial,
- qu'exécuter un **COMMIT** ou un **ROLLBACK** n'ont plus aucun effet, car nous sommes au début d'une nouvelle transaction (une session peut générer plusieurs transactions mais exécutées en série, voir le cours).
- pour la transaction T_2 tout se passe comme si T_1 n'avait pas existé.

  Recommencez la transaction T_1 et validez, cette fois, les mises à jour (*COMMIT*). Vous devriez constatez que :

  – il n'est pas possible d'annuler par un **ROLLBACK**. Une fois les données validées, elles sont pérennes. Le fait de ne pas pouvoir annuler correspond à la notion de durabilité. Autrement dit, un commit valide les données de manière définitives. Après une validation, les données intégre l'état de la base (nous appelons état de la base à un instant t l'ensemble des données validées).

  Maintenant, la transaction T_2 voit les mises à jour de T_1, et on en déduit que le niveau d'isolation par défaut d'Oracle est celui de *READ COMMITTED*. Par contre dans le cas où le niveau d'isolation est *REPEATABLE READ*[3], T_2 ne verra pas ces mises à jour, car elle a commencé avant T_1, et continue à accéder à l'état de la base au moment elle a commencé son exécution.

[2] Ceci permet d'assurer la cohérence des données
[3] les mêmes lectures renvoient toujours le même résultat

isolation incomplète = incohérence possible

Réinitialisez, toujours en mode *READ COMMITTED*. Maintenant déroulez deux transactions T_1 (réservation de 2 billets pour c_1) et T_2 (réservation de 3 billets pour c_2) en parallèle selon l'alternance suivante :

- on effectue les deux sélections pour T_1 (vol et client) ;
- on effectue les deux sélections pour T_2 (vol et client) ;
- on effectue les deux mises à jour de T_1, et on valide ;
- on effectue les deux mises à jour de T_2, et on valide

L'objectif de cette manipulation est de reproduire le problème des mises à jour perdues, dans le cas où le niveau d'isolation n'est pas maximale. Une fois les deux transactions terminées, vous devriez constater que les deux clients ont bien réservé 2+3 = 5 billets, mais que seulement 3 billets ont été réservés pour le vol.

La base se retrouve dans un état incohérent. Chaque transaction, vue individuellement, est correcte. Il y a donc un défaut de concurrence dû à un niveau d'isolation incomplet.

isolation complète = blocage et rejet des transactions possibles

Pour assurer une cohérence complète, il faut choisir le mode d'isolation complet (serializable). Réinitialisez les deux sessions, en choisissant ce mode, et refaites la même exécution concurrente précédente. Cette fois, une des deux transaction sera rejetée.

Reproduire l'exemple suivant d'une exécution concurrente, vu en cours :

$$r_1(d)w_2(d)w_2(d')C_2w_1(d')C_1$$

Conclure : l'algorithme implémenter dans ORACLE est-il le même que celui vu en cours (verrouillage à deux phases) ?.