V- Langage de Contrôle des Transactions (TCL)

I- Problématique des accès concurrents

Dans la majorité des développements informatiques se pose la question des accès simultanés à une donnée par plusieurs utilisateurs différents.

En effet, un développeur d'application doit prévoir la gestion des accès concurrents en utilisant les outils fournis par la base de données.

La majorité des SGBDR autorisent la réservation de données avant mise à jour afin d'empêcher un autre utilisateur de modifier cette même donnée avant que le premier n'ait validé sa modification.

1. Cas des accès concurrents

a. mises à jour simultanées

b. incohérence des données suite à une modification d'un autre utilisateur

Un utilisateur lit la table Tarifs au début de son programme, puis réalise des traitements par rapport à ce qu'il a récupéré.

Il relit cette même table en fin de traitement pour récupérer les mêmes informations. Entre-temps, un autre utilisateur a réalisé une modification ou une insertion dans la table.

L'utilisateur 1 doit verrouiller l'enregistrement afin que personne ne puisse le modifier pendant le passage de son programme.

Ce phénomène est appelé : **lecture non répétitive**.

2. Le mécanisme de verrouillage

Les mécanismes de verrouillage existent dans les SGBDR pour permettre aux développeurs de sécuriser leurs mises à jour et garantir une cohérence de la base.

II-Notion de transaction

Afin de limiter les problématiques indiquées dans les paragraphes précédents, il faut que le développeur se pose la question : quels sont les objets (ligne, colonne, table…) que je manipule dans mon traitement et comment éviter qu'un

autre utilisateur ne puisse les atteindre avant que j'aie terminé mes mises à jour ?

À ce moment, on commence à parler de « transaction ».

1. Définition d'une transaction

La transaction permet de borner le début et la fin d'une action dans la base, sur une ou plusieurs tables, qui doivent rester cohérentes.

La transaction est une notion qui vient des applications dites « transactionnelles ». À chaque fois qu'il existe un dialogue écran entre l'application et l'utilisateur, on parle d'application transactionnelle.

Toutes les applications utilisent des mécanismes de verrouillage.

2. Comment éviter les incohérences de données

Plusieurs méthodes existent pour garantir une cohérence dans la base :

- Lancer les transactions les unes derrière les autres (en série). L'inconvénient principal est le temps d'attente très important pour les utilisateurs. Cela revient à avoir une application mono-utilisateur !
- Possibilité de bloquer en début de programme principal tous les objets implicitement puis les libérer à la fin. Cette méthode risque de bloquer les autres utilisateurs trop longtemps si l'utilisateur ne valide pas rapidement son écran et part en réunion quelque temps.
- Pas de verrouillage des enregistrements en début mais lecture des données puis sauvegarde en mémoire de leur contenu puis relecture juste avant la mise à jour pour s'assurer que les données n'ont pas été modifiées entre temps. Si les valeurs sauvegardées sont différentes des valeurs relues, alors abandon de la transaction et affichage d'un message à l'utilisateur pour signaler le problème et lui indiquer qu'il doit ressaisir les éléments.

3. Mise en œuvre d'un verrouillage

Selon la norme SQL-92, on peut indiquer à la base de données quel type de méthode on veut utiliser.

Pour activer un mode de verrouillage, il faut utiliser la commande : SET TRANSACTION ISOLATION LEVEL.

Il existe quatre modes de verrouillage.

Le mode par défaut dans MySQL est le mode REPEATABLE-READ. Pour Oracle, PostgreSQL et SQL Server, il s'agit du mode READ COMMITTED.

Généralement, l'action s'applique à une session, celle de l'utilisateur qui l'active, certains SGBDR permettent d'appliquer le niveau de verrouillage à toutes les sessions en cours en utilisant le mot GLOBAL.

Nous allons décrire les différents modes.

a. READ UNCOMMITTED

Quand ce mode est activé, l'utilisateur a la possibilité de visualiser les lignes qui sont en cours de modification par une autre personne, même si elles ne sont pas validées. Les données ne sont donc pas garanties du point de vue fonctionnel, en effet tant que l'autre utilisateur n'a pas validé sa modification les données sont volatiles.

b. READ COMMITTED

L'utilisateur ne visualise que les données validées. En aucun cas il ne peut voir des lignes en cours de modification non validées. C'est le mode de fonctionnement par défaut utilisé par Oracle. Ce mode garantit une cohérence des données entre les utilisateurs.

c. REPEATABLE-READ

Toutes les données utilisées par une requête sont verrouillées, ceci empêche toute modification de données par les autres utilisateurs. En revanche, les autres utilisateurs ont encore la possibilité d'insérer de nouvelles lignes dans la table. Ce niveau est assez restrictif et peut provoquer des attentes importantes auprès des utilisateurs.

d. SERIALIZABLE

C'est le mode le plus restrictif et le plus contraignant pour les utilisateurs. En contrepartie, c'est le mode le plus sécurisant pour le développeur mais il est déconseillé de l'utiliser sauf sur des applications avec très peu d'utilisateurs. Ce mode bloque l'ensemble de la table à chaque demande de l'utilisateur. Aucune action n'est possible par les autres utilisateurs tant que le premier n'a pas validé sa transaction.

e. Syntaxes

Syntaxe MySQL

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
```

```
{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE
};
```

Syntaxe Oracle

```
SET TRANSACTION ISOLATION LEVEL
{ READ COMMITTED
| SERIALIZABLE
};
```

Syntaxe SQL Server et PostgreSQL

```
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE
};
```

Oracle propose d'autres possibilités, notamment avec le READ ONLY qui permet d'indiquer que l'on ne fait que de la lecture ou le READ WRITE pour indiquer que l'on fait de la mise à jour.

4. Mise en œuvre d'un verrouillage applicatif

Les règles de verrouillage qui s'appliquent pour les transactions sont mises en œuvre la plupart du temps par les administrateurs de la base et s'appliquent à l'ensemble de développeurs.

Indépendamment des règles gérées au niveau d'une transaction comme indiqué dans la section précédente, il est possible de bloquer des lignes d'une table pour éviter qu'un autre utilisateur les modifie avant la fin de notre traitement.

Il existe deux méthodes principales pour gérer ces cas. Le verrou exclusif (FOR UPDATE) qui bloque toutes les lignes ramenées par le SELECT, empêchant toute

modification d'un autre utilisateur. Le verrou partagé (SHARE MODE) qui laisse la possibilité aux autres utilisateurs de visualiser les lignes bloquées dans leur état avant la pose du verrou.

Syntaxe Oracle et PostgreSQL

```
SELECT ... FROM .... WHERE ....
FOR UPDATE;
```

ou :

```
LOCK TABLE <nom table> IN SHARE MODE;
```

Syntaxe MySQL

```
SELECT ... FROM .... WHERE ....
FOR UPDATE;
```

ou :

```
SELECT ... FROM .... WHERE ....
LOCK IN SHARE MODE;
```

Syntaxe SQL Server

```
SELECT ... FROM .... WHERE ....
WITH (NOWAIT) ; --> affiche un message à la place du résultat
SELECT ... FROM .... WHERE ....
WITH (READPAST); --> n'affiche pas les lignes verrouillées
SELECT ... FROM .... WHERE ....
WITH (NOLOCK) ; --> affiche les lignes verrouillées dirtyread
```

Exemple Oracle et PostgreSQL

```
SELECT idTarif, hotel, typeChambre, DateDebut, DateFin, Prix FROM
Tarifs WHERE hotel = 1
FOR UPDATE;
```

ou :

```
LOCK TABLE Tarifs IN SHARE MODE;
```

Exemple MySQL

```
SELECT idTarif, hotel, typeChambre, DateDebut, DateFin, Prix FROM
Tarifs WHERE hotel = 1
LOCK IN SHARE MODE;
```

Exemple SQL Server

```
select * from Tarifs with (NOWAIT);
select * from Tarifs with (READPAST);
select * from Tarifs with (NOLOCK);
```

5. Validation des modifications (COMMIT)

Le principe de base dans tous les SGBDR est de laisser à l'utilisateur le soin de valider ses modifications à la fin de sa transaction et ainsi de rendre visible les changements réalisés dans les données aux autres utilisateurs.

Celui-ci peut être programmé par le développeur au moment souhaité afin de garantir l'intégrité référentielle de la base de données. C'est donc lié au fonctionnement même de chaque programme.

Le COMMIT valide toutes les modifications, insertions ou suppressions qui ont été réalisées depuis le dernier COMMIT ou depuis le début de la transaction.

Dans un programme, on peut mettre plusieurs COMMIT permettant ainsi de découper le traitement en séquences cohérentes fonctionnellement.

Le COMMIT permet également de libérer tous les verrous qui auraient été posés auparavant.

Syntaxe

```
COMMIT;
```

6. Abandon des modifications (ROLLBACK)

Le ROLLBACK est le contraire du COMMIT. Celui-ci invalide toutes les modifications, insertions et suppressions qui auraient été réalisées depuis le dernier COMMIT ou le début de la transaction ou depuis le dernier point de synchronisation (SAVEPOINT).

C'est très utile lorsque l'on réalise des essais d'ordres SQL sur une base et que l'on fait de fausses manipulations. Dans ce cas avant de quitter, il faut lancer le ROLLBACK et la base revient dans son état initial.

Syntaxe Oracle, MySQL et PostgreSQL

```
ROLLBACK [TO SAVEPOINT <nom du point de contrôle>];
```

Syntaxe SQL Server

```
ROLLBACK [ TRANSACTION <nom du point de contrôle>];
```

7. Les points de synchronisation (SAVEPOINT)

Lorsque l'on déroule un ensemble d'ordre SQL visant à valider un traitement fonctionnel, il existe la possibilité de poser des points de contrôle à plusieurs stades, permettant ainsi en cas de problème de revenir à un point cohérent de la base.

À noter qu'un COMMIT annule tous les points de synchronisation. Un ROLLBACK sans point de synchronisation annule également tous les points de synchronisation et remet la base dans l'état initial.

Syntaxe Oracle, MySQL et PostgreSQL

```
SAVEPOINT <nom du point de contrôle>;
```

Exemple

```
SAVEPOINT AVANT_INSERT;
```

Syntaxe SQL Server

```
SAVE TRANSACTION <nom du point de contrôle>;
```

Exemple

```
SAVE TRANSACTION AVANT_INSERT;
```