LES TRANSACTIONS

1. Introduction

Un SGBDR permet à plusieurs clients d'utiliser simultanément la même base de données. Ainsi un client peut consulter des données qu'un autre client est en train de modifier. Mais que se passe-t-il lorsque deux clients tentent de modifier au même moment les mêmes données? Pour éviter les problèmes (c'est-à-dire avoir une base avec des données incohérentes) le mécanisme des transactions a été inventé.

Une **transaction** permet de définir l'atomicité d'un traitement. Un traitement atomique s'exécute en mode « tout ou rien » : soit toutes les opérations relatives au traitement sont réalisées, soit elles sont toutes annulées. Mais l'atomicité n'est pas une condition suffisante. Une transaction idéale doit satisfaire les quatre critères suivants :

- **A**tomicité. Les opérations du traitement considéré ne doivent pas être interrompues.
- **C**ohérence. Le résultat ou les changements induits par une transaction doivent impérativement préserver la cohérence de la base de données.
- **I**solation. Les transactions sont isolées les unes des autres. Les effets d'une transaction ne peuvent être « vus » pas les autres transactions.
- **D**urabilité. Une fois la transaction validée, ses effets doivent perdurer c'est-à-dire que les données sont persistantes (inscrites de manière permanentes) même s'il s'ensuit une défaillance dans le système.

L'acronyme ACID est constitué par les quatre initiales de ces propriétés et désigne un système de gestion de base de données relationnelles offrant un parfait mécanisme de transaction.

Plus précisément, une transaction est une unité d'interaction avec un système de gestion de base de données, qui est traité d'une manière cohérente et fiable indépendamment des autres transactions. Une transaction unique peut regrouper plusieurs requêtes, chacune avec des ordres de lecture ou d'écriture dans la base de données. Par exemple après avoir créé un enregistrement la commande COMMIT permet de valider cette action. A contrario si après une commande de suppression de données vous voulez annulez la dernière transaction vous taperez la commande ROLLBACK. Vous récupérerez ainsi les données effacées.

Pour commencer une transaction :
`begin`

Pour annuler une transaction :
`rollback`

Pour valider une transaction :
`commit` ou `end`

Remarque : par défaut *psql* (tout comme Squirrel) est en mode *autocommit* c'est-à-dire que *psql* effectue un *commit* après chaque commande. Pour neutraliser ce comportement exécuter la commande (en respectant la casse et les espaces) **`\set AUTOCOMMIT 'off'`** . Pour visualiser les variables internes, entrer la commande **`\set`**.

2. Un cas concret

Dans la base *magasin_0_5*, on veut écrire des requêtes qui insèrent les données relatives à une commande qui porte sur dix produits.

On veut avoir la garantie que l'opération se déroule soit en totalité soit que la commande ne soit pas du tout insérée. On veut donc être absolument sûr que les données n'ont pas été insérées en partie par exemple à cause d'une panne du serveur.

Pour réaliser cela, il faut :

- démarrer une transaction (*begin*) ;
- faire une insertion dans la table *commande* ;
- faire dix insertions dans la table *porter* ;
- terminer la transaction (*commit*).

3. Exemples avec deux sessions simultanées

3.1. Consultation et modification simultanées

On introduit d'abord un nouveau client qui va nous permettre d'expérimenter :

```
insert into client(id, nom, prenom, datenaissance, sexe, parrain) values
(99999,'essai', 'a', '2007-02-15', true, null);
```

Démarrez deux transactions avec deux clients *psql* différents et exécutez les requêtes exactement dans l'ordre spécifié. Dans l'une on va observer un enregistrement que l'on aura modifié dans la seconde.

1ère session	***2nde session***
1) `begin;`	
	2) `begin;`
3) `select * from client where id=99999;`	
	4) `select * from client where id=99999;`
	5) `update client set prenom='toto' where id=99999;`
	6) `select * from client where id=99999;`
7) `select * from client where id=99999;`	
	8) `end;`
	9) `select * from client where id=99999;`
10) `select * from client where id=99999;`	
11) `end;`	
12) `select * from client where id=99999;`	

3.2. Deux modifications simultanées

On introduit un nouveau client :

```
insert into client(id, nom, prenom, datenaissance, sexe, parrain) values
(99999,'essai', 'a', '2007-02-15', true, null);
```

Démarrez deux transactions avec deux clients psql différents. Dans l'une on va tenter de mettre à jour des enregistrements que l'on aura supprimés dans la seconde.

1ère session	*2nde session*
1) `begin;`	
	2) `begin;`
3) `select * from client where id=99999;`	
	4) `delete from client where id=99999;`
5) `select * from client where id=99999;`	
6) `update client set prenom='titi' where id=99999;` *=> attente*	
	7) `end;`
déblocage => aucune lignes mise à jour	
8) `end;`	

Pour mettre en place ces mécanismes, Postgresql utilise des verrous que l'on peut observer dans la table système `pg_locks`.

```
select * from pg_locks ;
```