Les transactions

Bases de Données

Année 2007-08

Généralités

une transaction : suite d'opérations sur la base qui ne conservent la cohérence des données que lorsqu'elles sont toutes effectuées.

Exemples :
- emprunt de livres Lorsqu'un adhérent rend un livre, on va supprimer l'enregistrement correspondant de la table `emprunt` pour créer un enregistrement dans la table `histoEmprunt`
- opérations bancaires Lors d'un virement de compte à compte, on doit débiter une somme sur un compte avant de la créditer sur un autre.

Généralités

Que se passe-t-il s'il arrive un problème entre les opérations ? D'autant que les problèmes peuvent être de nature différentes :
- panne logicielle (rupture du réseau ...)
- incohérence des données (le premier compte ne contient même pas la somme à débiter...)

Nécessité de préciser que les opérations appartiennent à une même transaction : la transaction complète réussit et toutes les opérations sont effectuée sur la base, ou bien la transaction échoue, et toutes les opérations sont annulées.

Généralités

Une transaction commence par le mot-clé BEGIN, ou BEGIN WORK,ou BEGIN TRANSACTION,ou START TRANSACTION, et termine
- soit par un succès : COMMIT
- soit par un échec : ROLLBACK

Exemples

```
loca=# begin ;
BEGIN
loca=# select couleur from voiture where id_voiture = 4 ;
 couleur
```

```
---------
 vert
loca=# update voiture set couleur = 'rose'
            where id_voiture = 4 ;
UPDATE 1
loca=# commit ;
COMMIT
loca=# select couleur from voiture where id_voiture = 4 ;
 couleur
---------
 rose
```

Exemples

```
loca=# begin ;
BEGIN
loca=# select * from voiture where id_voiture = 4 ;
 id_voiture | num_immatriculation | id_modele | couleur
------------+---------------------+-----------+---------
          4 | 2345 FG 78          |         1 | rose
loca=# update voiture set couleur = mauve'
       where id_voiture = 4 ;
UPDATE 1
loca=# rollback ;
ROLLBACK
loca=# select * from voiture where id_voiture = 4 ;
 id_voiture | num_immatriculation | id_modele | couleur
------------+---------------------+-----------+---------
          4 | 2345 FG 78          |         1 | rose
```

Exemples

Une erreur annule une transaction. . .

Exemples

```
location=# begin ;
BEGIN
location=# select couleur from voiture
where id_voiture = 4 ;
 rose
location=# update voiture set couleur = 'rose à pois bleus' ;
ERROR :  value too long for type character varying(10)
location=# update voiture set couleur = 'grise' ;
ERROR :  current transaction is aborted,
```

```
queries ignored until end of transaction block
location=# commit ;
COMMIT
location=# select couleur from voiture
where id_voiture = 4 ;
 rose
```

Généralités

Quelques remarques :

- ne pas confondre BEGIN qui démarre une transaction et BEGIN le mot-clé PLSQL qui indique le début d'une fonction ;
- pas de transactions imbriquées en PostgreSQL ;
- de ce fait, pas de COMMIT ou de ROLLBACK à l'intérieur d'une fonction ;
- jusqu'à présent : travail avec l'option AUTOCOMMIT. Chaque instruction SQL est une transaction.

Généralités

Dans la base `maPetiteEntreprise` vue en interro :

```
commerce=#begin ;
BEGIN
commerce=#select creeFacture(3) ;
 creeFacture
------------

(1 ligne)

commerce=# commit ;
COMMIT
```

Vers les accès concurrents

- Les étapes intermédiaires d'une transaction ne sont pas visibles par les autres utilisateurs.

Vers les accès concurrents

```
loca=# begin ;
loca=# update voiture
set couleur='vert' where id_voiture=4 ;
UPDATE 1
loca=#                    | loca=# select couleur
                          | from voiture where
                          | id_voiture=4 ;
                          |  couleur
```

```
                          | ---------
                          |  rose
loca=#commit ;            |
COMMIT                    |
loca=#                    | loca=# select couleur
                          | from voiture where id_voiture=4 ;
                          |  couleur
                          | ---------
                          |  vert
```

Les accès concurrents

Quelle vision des données ?

Effets des accès concurrents sur la lecture de données :

- lecture sale : une transaction lit des données écrites par une transaction concurrente non validée ;
- lecture non reproductible : une transaction relit des données et obtient des données modifiées (à cause d'une transaction concurrente validée) ;
- lecture d'enregistrements fantômes : une transaction re-exécute une requête et obtient un ensemble d'enregistrements différents (à cause d'une transaction concurrente validée)

Les accès concurrents

Le standard SQL définit 4 niveaux d'isolation des transactions.

Niveau d'isolation	**lect. sale**	**lec. non repr.**	**lect. fant.**
read uncommited	possible	possible	possible
read commited	impossible	possible	possible
repeatable read	impossible	impossible	possible
serializable	impossible	impossible	impossible

PostgreSQL fournit les niveaux read commited et serializable.

Les accès concurrents

Chaque session concurrente voit un cliché de la base. Le résultat des requêtes dépend donc du moment où est pris le cliché.

- read commited :
  - cliché pris en début de chaque requête, sur des modifications validées
  - cas d'une requête R UPDATE ou DELETE : si une transaction B concurrente accède en modification les mêmes enregistrements alors R bloquée jusqu'à validation ou annulation de B
  - conditions de recherche sont ré-évaluées pour les enregistrements satisfaisant R : R effectuée pour ceux satisfaisant toujours les critères

Les accès concurrents

Exemple avec deux UPDATE concurrents.

Les accès concurrents

```
temps | session A                | session B
t     | ex=# begin ;              | ex=# begin ;
      | BEGIN                    | BEGIN
t+1   | ex=# update voiture       |
      |      set couleur='rose' ; |
      |      where id_voiture=1 ;|
t+2   | UPDATE 1                  |
t+3   |                           | ex=# update voiture
      |                           |      set nom = 'truc'
      |                           |      where id_voiture=1 ;
...   |                           | -- requête bloquée ! !
t+10  | ex=# commit ;             |
t+11  | COMMIT                    | UPDATE 1
t+12  |                           | ex=# commit ;
t+13  |                           | COMMIT
```

Les accès concurrents

Exemple avec un INSERT et un UPDATE concurrents.

Les accès concurrents

```
temps | session A                | session B
t     | ex=# begin ;              | ex=# begin ;
      | BEGIN                    | BEGIN
t+1   |                           | ex#= select count(id_voiture)
      |                           |      from voiture
t+2   | ex=# insert into voiture|      where couleur = 'rose' ;
      |      (couleur)           |  count
      |      values('rose') ;    | ------
      | INSERT 1                 |  3
t+3   |                           | ex=# update voiture
      |                           |      set couleur = 'mauve'
      |                           |      where couleur = 'rose'
t+4   | ex=# commit ;             | UPDATE 3
t+5   | COMMIT                    |
t+6   |                           | ex=# commit ;
t+7   |                           | COMMIT
```

Les accès concurrents

Exemple avec un UPDATE et un DELETE concurrents.

Les accès concurrents

```
temps | session A                | session B
t     | ex=# begin;              | ex=# begin;
      | BEGIN                    | BEGIN
t+1   | ex=# select couleur from | ex#= select count(id_voiture)
      |      voiture where       |      from voiture
      |      id_voiture = 3;     |      where couleur = 'rose';
t+2   |  couleur                 |  count
      | --------                 | ------
      |  rose                    | 3
t+3   | ex=# delete from voiture |
t+4   |      where id_voiture=3; | ex=# update voiture
      | DELETE 1                 |      set couleur = 'mauve'
      |                          |      where couleur = 'rose'
t+5   | ex=# commit;             |   -- requête bloquée!!
t+6   | COMMIT                   | UPDATE 2
t+7   |                          | ex=# commit;
t+8   |                          | COMMIT
```

Les accès concurrents

Attention : au niveau d'isolation read committed, comme deux SELECT qui se suivent ne voient pas la même version de la base, cela peut donner des incohérences au niveau de la transaction.

Incohérences au niveau read commited

```
temps | session A                | session B
t     | ex=# begin;              | ex=# begin; -- et
      | BEGIN                    | BEGIN -- entrée ds une fct plpgsql
t+1   | ex=# select couleur      | ex#= select count(id_voiture)
      |   from voiture           |      into nb1
      |   where id_voiture=3;    |      from voiture
      |   couleur                |      where couleur = 'rose';
t+2   |   -------                |  count
      |   rose                   | ------
      |                          |  3
```

Incohérences au niveau read commited

```
      | ex=# delete from voiture |
t+3   | where id_voiture=3;      |
t+4   | DELETE 1                 | ex=# select count(*) into nb2
      |                          |      from voiture;
```

```
      |                          |  count
t+5   | ex=# commit ;            | ------
t+6   | COMMIT                   | 10
      |                          | ex=# return nb1/nb2 ;
t+7   |                          | ex=# commit ;
t+8   |                          | COMMIT
```

Les accès concurrents

- Serializable : le niveau le plus strict d'isolation.
  - cliché de la base pris au début de la première requête de la transaction
  - conservé pour toute la durée de la transaction
  - ordre UPDATE ou DELETE : si modification des enregistrements par une transaction concurrente, la transaction échoue
  - $\Longrightarrow$ nécessité de retenter des transactions échouées
  - transactions qui n'effectuent que des lectures n'ont jamais de conflits, et proposent une vue complètement cohérente sur toute la transaction.

Les verrous

Pour le moment, on reste dans le principe

- une écriture ne bloque pas une lecture
- une lecture ne bloque pas une écriture

Pas toujours satisfaisant...

- DROP TABLE ne peut pas être exécuté en concurrence avec d'autres opérations ;
- base maPetiteEntreprise création d'une facture ;

Les verrous

On ajoute des verrous, implicitement ou explicitement, sur les tables ou les enregistrements.

- un verrou peut être posé implicitement (par un ordre SQL)
- ou explicitement (ordre LOCK TABLE [IN `verrou_mode` MODE])
- un verrou est acquis jusqu'à la fin de la transaction (limitation Postgres)

Les verrous

- deux types de verrous : sur une table complète, ou sur des enregistrements d'une table
- hiérarchie sur les verrous (de très contraignant à très permissif)
- la priorité sur les verrous est donnée suivant leur ordre chronologique
- les noms sont historiques !
- clé :
  - exclusive : empêche le même type de verrou d'être posé en même temps ;
  - share : accepte la pose d'un même type de verrou

Les verrous de table

Du plus exclusif au plus permissif. Tous ces verrous peuvent être acquis de manière explicite.

- ACCESS EXCLUSIVE conflit avec tous les autres modes. Acquis par DROP TABLE, ALTER TABLE, VACUUM FULL, ou bien LOCK TABLE sans précisions !
- EXCLUSIVE accepte seulement les verrous ACCESS SHARE, i.e. les SELECT. N'est acquis implicitement par aucun ordre SQL

Les verrous de table

- SHARE ROW EXCLUSIVE accepte les accès concurrents en consultation uniquement (SELECT avec ou sans clause FOR UPDATE)(conflit avec tous les autres modes). N'est acquis implicitement par aucun ordre SQL
- SHARE acquis par CREATE INDEX, idem que le précédent mais non exclusif ;
- SHARE UPDATE EXCLUSIVE acquis par VACUUM, conflit avec les quatre verrous précédents, et lui-même : protège contre les VACUUM concurrents et les modifications de schémas de table (ALTER TABLE)

Les verrous de table

- ROW EXCLUSIVE acquis par UPDATE, DELETE, INSERT, conflit avec les quatre premiers verrous, et lui-même
- ROW SHARE acquis par SELECT FOR UPDATE, conflit avec les deux premiers verrous (ACCESS EXCLUSIVE et EXCLUSIVE)
- ACCESS SHARE acquis par SELECT, conflit avec le premier verrou (ACCESS EXCLUSIVE)

Les verrous d'enregistrements

Limitent l'effet du verrouillage au niveau des enregistrements accédés, pas de la table complète.

- acquis par une modification, une suppression ou une sélection pour mise-à-jour (UPDATE, DELETE, SELECT FOR UPDATE) ;
- jusqu'à la fin de la transaction ;
- ne bloquent pas la lecture (SELECT)
- bloquent l'écriture sur ces enregistrements
- SELECT FOR UPDATE permet de poser un tel verrou sans modifier les données

Cas d'utilisations

On veut, pour une transaction, consulter des données sans qu'elles ne soient modifiées en cours de route.

$\Longrightarrow$ verrou en mode SHARE

Refuse toute écriture, accepte d'autres verrous en mode SHARE

- si une écriture est en cours . . .on est bloqué !
- lorsqu'on acquiert le verrou, on empêche toute mise-à-jour jusqu'à la fin de la transaction

Cas d'utilisations

```
Temps | Session A           | Session B
t     | bd=#begin ;          | bd=#begin ;
      | BEGIN                | BEGIN
t+1   | bd=#update XXX ;     |
      | UPDATE               |
t+2   |                      | bd=#lock table XXX
      |                      |     in share mode ;
      |                      | -- requête bloquée
t+4   | bd=# commit ;        |
      | COMMIT               | LOCK TABLE
t+5   | bd=# update XXX ;    | bd=# select ...
      | -- requête bloquée   |
```

Les deadlocks

Même cas de figure, mais on en profite pour mettre à jour les données.

```
Temps | Session A            | Session B
t     | bd=#begin ;          | bd=#begin ;
      | BEGIN                | BEGIN
t+1   | bd=#lock table XXX   | bd=#lock table XXX
      |     in share mode ;  |     in share mode ;
      | LOCK TABLE           | LOCK TABLE
      |                      |
t+2   | bd=# select XXX... ; | bd=# select XXX ... ;
      | -- OK !              | -- OK !
t+3   | bd=# update XXX ;    |
      | -- requête bloquée ! |
t+4   |                      | bd=# update XXX ... ;
      |                      | -- erreur ! deadlock détecté
t+5   | UPDATE               | -- verrou levé...
```

Cas d'utilisations

- il fallait poser un verrou en mode SHARE ROW EXCLUSIVE
- en cas de deadlock, on ne peut pas prédire quel verrou sera levé
- est-ce une bonne idée de poser un verrou restrictif dans une transaction qui attend des réponses de l'utilisateur ?
- on veut faire une insertion sur une table après avoir récupéré la valeur d'une clé étrangère qu'on doit utiliser dans l'insertion. Doit-on utiliser un verrou, et si oui, lequel ?

Cas d'utilisations

Les verrous les plus fréquemment utilisés (explicitement) :

- SHARE MODE pour des lectures cohérentes ;
- SHARE ROW EXCLUSIVE MODE pour des lectures cohérentes sans souci de deadlock pour l'écriture de données