SQL – Transactions

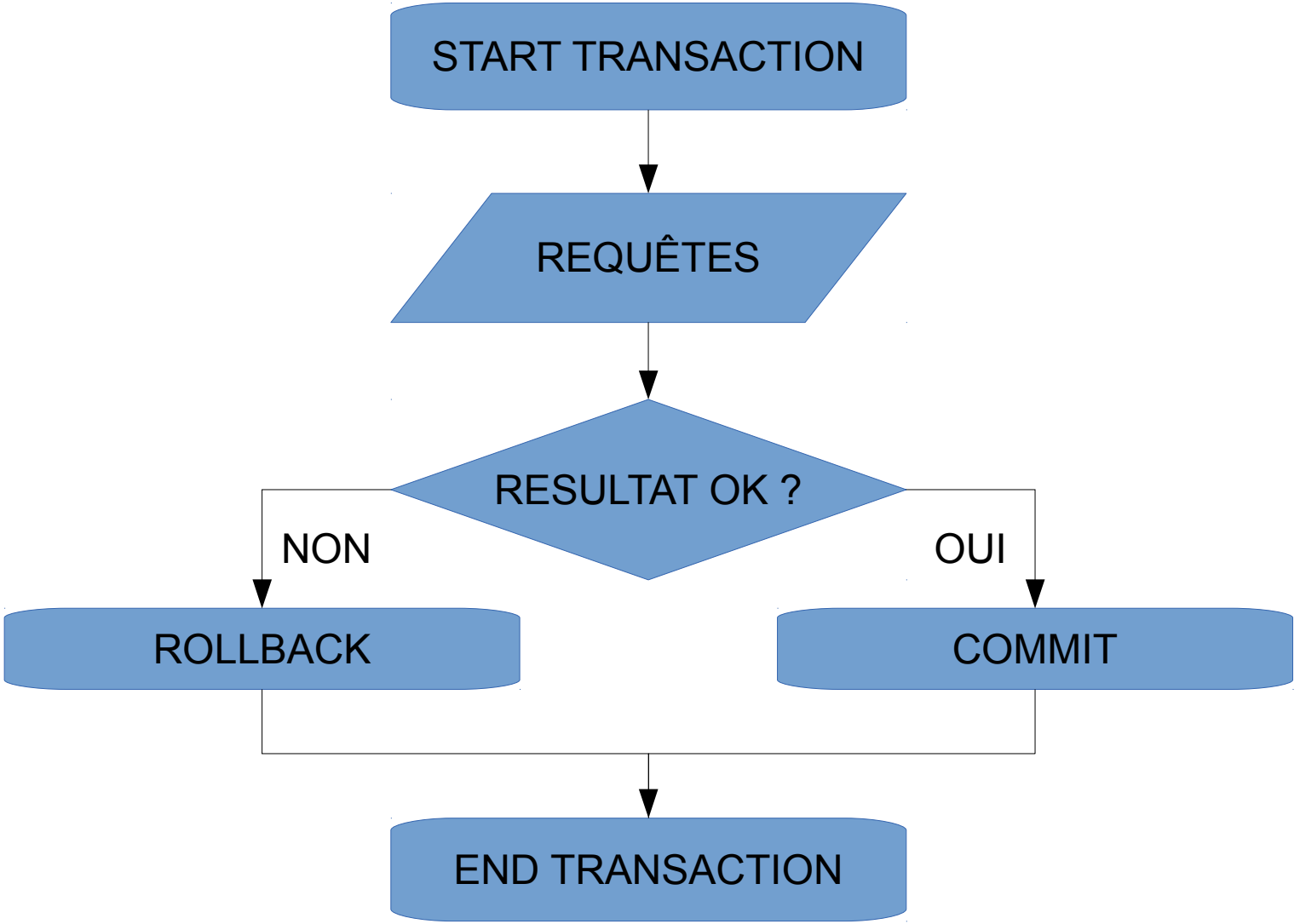
Afin de rendre les enchaînements de requêtes plus sûrs mais surtout plus déterministes, nous avons besoin d'un nouveau mécanisme : les transactions

SQL – Transactions

Le principe est d'encapsuler une série de requêtes dans un bloc.

Nous veillerons à ce que ce bloc soit complètement exécuté, sinon, il peut être annulé.

SQL – Transactions



SQL – Transactions

- Concrètement, le test de résultat est réalisé après chaque requête
- `COMMIT` sert à valider les requêtes
- `ROLLBACK` sert à annuler les requêtes
- Le mode transaction n'est possible en MySQL qu'avec les tables qui utilisent le moteur InnoDB

SQL – Transactions

- MySQL étant par défaut configuré pour valider automatiquement les requêtes, on doit commencer par modifier ce comportement
- Affichage du mode en cours :

```
SHOW variables WHERE variable_name LIKE '%autocommit%';
-- ou
SELECT @@autocommit;
```

- Modification du mode (pour la durée de la session) :

```
SET autocommit=OFF;
```

SQL – Transactions

- Dès que le mode `autocommit` est désactivé, une transaction est démarrée
- Elle dure jusqu'à la fin de la session ou jusqu'à l'exécution d'une commande `commit` ou `rollback`
- On peut démarrer explicitement une transaction avec la requête suivante (indépendant du mode `autocommit`) :

```
START TRANSACTION; -- SQL2
-- ou
BEGIN; -- MySQL
-- ou
BEGIN WORK; -- MySQL
```

Rem : `START TRANSACTION` désactive l'`autocommit` jusqu'à la fin de celle-ci

SQL – Transactions

Remarques

- Lorsqu'une transaction démarre, elle travaille sur une copie des données, pas sur les données réelles. Si la transaction est validée, les données de celle-ci remplace les données réelles.

- Tous les types de requêtes ne peuvent pas être annulés, il s'agit notamment des requêtes DDL.

SQL – Transactions

Exemple 1

- Dans une transaction, on insère un nouveau client
- On sélectionne tous les clients
- On fait un `ROLLBACK`
- On sélectionne tous les clients

SQL – Transactions

Exemple 2

- Dans une transaction, on efface un client
- On sélectionne tous les clients
- On fait un `ROLLBACK`
- On sélectionne tous les clients

Transactions – Jalons

On ne peut pas imbriquer les transactions.

En revanche, il est possible d'ajouter des repères intermédiaires.

Ces jalons permettent de ne pas tout annuler en cas d'échec d'une requête.

Transactions – Jalons

Ajouter un jalon :

```
SAVEPOINT nom_du_jalon;
```

Retirer un jalon :

```
RELEASE SAVEPOINT nom_du_jalon;
```

Utiliser un jalon :

```
ROLLBACK TO [SAVEPOINT] nom_du_jalon;
```

Annule toutes les requêtes comprises entre cette commande et le jalon

Transactions – Jalons

Exemple (faire des sélections intermédiaires pour suivre les modifications)

```
START TRANSACTION;

INSERT INTO client VALUES(NULL, 'NOM5', 'Prénom5',
'0, rue du test 5', 'Test', 9999, NULL, 0);

SAVEPOINT jalon_delete;

DELETE FROM client WHERE nom='FRANCK';

ROLLBACK TO jalon_delete;

COMMIT;
```

Transactions – Jalons

À tester :

Même exemple que précédemment, mais en réalisant des requêtes intermédiaires à partir d'un autre client

Transactions – ACID

Derrière cet acronyme se cache les caractéristiques qu'un SGBDR qui implémente les transactions doit respecter :

- A : Atomicité
- C : Cohérence
- I : Isolation
- D : Durabilité

Transactions – ACID

A : Atomicité

Toutes les requêtes d'une transaction constituent un groupe indivisible : soit tout est exécuté, soit rien ne l'est.

- `autocommit`
- `COMMIT`
- `ROLLBACK`

Transactions – ACID

C : Cohérence

Les données doivent rester cohérentes avant et après la transaction. Les états intermédiaires ne sont visibles qu'au sein de la transaction (voir exemple précédent)

- `doublewrite buffer` : écriture des données dans un buffer intermédiaire avant de les sauvegarder dans les fichiers de données MySQL
- `crash recovery` : grâce à un fichier de journalisation (redo log), il est possible de « rejouer » des transactions en cours de « `COMMIT` » qui auraient été interrompues avant sauvegarde dans le fichier de données MySQL

Transactions – ACID

I : Isolation ou Indépendance

- Une transaction doit rester indépendante des autres transactions
- Il existe plusieurs niveaux d'isolation qui seront vus à la fin de ce chapitre

Transactions – ACID

Exemple

Dans deux sessions différentes (clients + transactions) :

1) Essayez de mettre à jour les comptes de deux clients différents

2) Essayez de mettre à jour le compte d'un même client

Transactions – ACID

D : Durabilité

Les données doivent être intègres en toutes circonstances, même après une panne qui surviendrait juste après un `COMMIT` par exemple.

Transactions – Verrous

Les verrous sont les compléments indispensables des transactions : ils permettent de restreindre ou de bloquer l'accès aux données durant une transaction.

Ils rendent également possible la gestion de la concurrence lors des accès aux données.

Rem : l'implémentation de la notion de verrou SQL2 est très dépendante du SGBDR, même si les principes sous-jacents sont pratiquement identiques.

Les exemples vus dans ce chapitre s'appliquent au moteur InnoDB de MySQL

Transactions – Verrous

Les données « lockées » peuvent être des tables complètes ou des sous-ensembles de ces tables : nous parlerons de verrous de table (MyISAM et InnoDB) et de ligne (InnoDB).

Il va de soi que nous devons privilégier les verrous de ligne s'appliquant aux plus petits sous-ensembles possibles afin de ne pas rendre la DB inutilisable car constamment bloquée par une transaction ou une autre.

Transactions – Verrous de table

Pour verrouiller une table, nous utiliserons la syntaxe suivante :

```
LOCK TABLES table1[, table2] {READ | WRITE};
```

- L'option `READ` permet dans la session courante d'accéder à la table en lecture, mais plus en écriture. Les autres sessions peuvent y accéder en lecture mais pas en écriture (sauf si accès non-concurrentiels). Les autres tables ne sont plus accessibles.
- L'option `WRITE` permet dans la session courante d'accéder à la table en lecture et en écriture. Les autres sessions ne peuvent plus y accéder, ni en lecture, ni en écriture.

Transactions – Verrous de table

Il est possible de verrouiller plusieurs tables en une seule requête en séparant les tables par des virgules.

Il n'est en revanche pas possible de réaliser plusieurs commande `LOCK` à la suite car un `LOCK` réalise toujours un déverrouillage (`UNLOCK`) implicite de toutes les tables.

Transactions – Verrous de table

Pour déverrouiller une table, nous utiliserons la syntaxe suivante :

```
UNLOCK TABLES;
```

Toutes les tables seront dès lors déverrouillées.

Transactions – Verrous de table

Attention, dans une session, si on verrouille une table, les autres ne sont plus accessibles, même en lecture.

Exemple :

```
SELECT * FROM client;
SELECT * FROM commande;

LOCK TABLES client READ;

SELECT * FROM client;
SELECT * FROM commande;

UNLOCK TABLES;

SELECT * FROM client;
SELECT * FROM commande;
```

Transactions – Verrous de table

Remarques :

- Un `START TRANSACTION` exécute un `UNLOCK`
- Un `LOCK` ou `UNLOCK` dans une transaction exécute un `COMMIT`

Transactions – Verrous de ligne

Il existe deux types de verrous de ligne :

- Partagés : équivalents aux verrous de table en mode `READ`
- Exclusifs : équivalents aux verrous de table en mode `WRITE`

Transactions – Verrous de ligne

Un verrou partagé est ajouté explicitement avec l'option suivante :

```
LOCK IN SHARE MODE
```

Cette option est à ajouter à la fin d'une requête de sélection.

Le but recherché est, dans une transaction, d'éviter une modification d'un (ou plusieurs) enregistrement entre une sélection et une modification de celui-ci (ceux-ci).

Les autres sessions peuvent accéder en lecture à ces lignes qui sont en revanche protégées en écriture.

Transactions – Verrous de ligne

Exemple

```
SELECT *
FROM client
WHERE localite='verviers'
LOCK IN SHARE MODE;
```

Transactions – Verrous de ligne

Un verrou exclusif est ajouté explicitement avec l'option suivante :

```
FOR UPDATE
```

Cette option est à ajouter à la fin d'une requête de sélection.

Le but recherché est, dans une transaction, d'éviter tout accès à un (ou plusieurs) enregistrement durant une modification.

Les autres sessions ne peuvent accéder ni en lecture ni en écriture à ces lignes ou alors en lecture mais à l'état des enregistrements avant le début de la transaction.

Transactions – Verrous de ligne

Exemple

```
SELECT *
FROM client
WHERE localite='verviers'
FOR UPDATE;
```

Transactions – Verrous de ligne

Au sein d'une transaction :

- Un `DELETE` ou un `UPDATE` place implicitement un verrou exclusif sur les lignes filtrées par la clause `WHERE` si un index est configuré, sur toute la table sinon
- Un `INSERT` place implicitement un verrou exclusif sur l'enregistrement en cours d'ajout

Transactions – Verrous de ligne

Exercice 1

1) Dans une session, démarrez une transaction

2) Mettez à jour les codes postaux des clients qui habitent Namur

3) Dans une seconde session, démarrez une transaction

4) Dans cette même session, sélectionnez ces clients afin de vérifier la mise à jour

5) Dans la session 2, essayez de sélectionner également les données avec un `SELECT` régulier, en verrou partagé et en verrou exclusif

6) Dans le session 1, validez la transaction

7) Recommencez le point 4

Transactions – Verrous de ligne

Exercice 2

1) Dans deux clients distincts, démarrez une transaction

2) Dans la session 1, sélectionnez les clients qui habitent Verviers avec un verrou partagé

3) Dans la session 2, essayez de sélectionner également les données avec un `SELECT` régulier, en verrou partagé et en verrou exclusif

4) Dans la session 2, essayez de mettre à jour des données des clients qui n'habitent pas Verviers

5) Dans la session 2, essayez de mettre à jour des données des clients qui habitent Verviers

6) Dans la session 2, essayez d'insérer un client qui n'habite pas Verviers

7) Dans la session 2, essayez d'insérer un client qui habite Verviers

Transactions – Verrous de ligne

Exercice 3

1) Dans deux clients distincts, démarrez une transaction

2) Dans la session 1, sélectionnez les clients qui habitent Verviers avec un verrou exclusif

3) Dans la session 2, essayez de sélectionner également les données avec un `SELECT` régulier, en verrou partagé et en verrou exclusif

4) Dans la session 2, essayez de mettre à jour des données des clients qui n'habitent pas Verviers

5) Dans la session 2, essayez de mettre à jour des données des clients qui habitent Verviers

6) Dans la session 2, essayez d'insérer un client qui n'habite pas Verviers

7) Dans la session 2, essayez d'insérer un client qui habite Verviers

Transactions – Verrous de ligne

Exercice 4

1) Créez un index sur la colonne localité de la table client

2) Reproduisez les exercices 2 et 3

Pour rappel :

```
SHOW index FROM client;
CREATE INDEX idx_client_localite ON client(localite);
```

Transactions – Niveaux d'isolation

Les requêtes que nous venons de tester repose sur le niveau d'isolation sélectionné.

La requête générique de configuration du niveau d'isolation est la suivante :

```
SET [GLOBAL|SESSION] TRANSACTION
ISOLATION LEVEL {READ UNCOMMITTED|READ COMMITTED|
REPEATABLE READ|SERIALIZABLE};
```

Transactions – Niveaux d'isolation

Une option permet de déterminer la portée du niveau d'isolation :

- `GLOBAL` : pour toutes les sessions (sauf les sessions en cours)
- `SESSION` : pour la session courante

Si on ne précise pas la portée, le niveau d'isolation portera sur la prochaine transaction de la session courante

Transactions – Niveaux d'isolation

Il existe quatre niveaux d'isolation en MySQL :

1) `READ UNCOMMITTED` : permet de visualiser des modifications réalisées et pas encore validées par d'autres sessions durant la transaction en cours. Risque de lecture de données incohérentes.

2) `READ COMMITTED` : permet de visualiser des modifications réalisées et validées par d'autres sessions durant la transaction en cours.

3) `REPEATABLE READ` : niveau par défaut. Une même requête générique répétée plusieurs fois de suite donnera toujours le même résultat même si des modifications sont réalisées dans d'autres sessions : lit toujours la capture des données. Meilleur compromis.

4) `SERIALIZABLE` : verrouille toute la table, ce qui a pour effet de bloquer toutes les autres transactions qui tenteraient d'accéder aux données concernées. Mode le plus « sûr », mais le plus pénalisant en termes de performances.

Transactions – Niveaux d'isolation

Exercice 5

Pour chaque niveau d'isolation, en global :

1) Dans deux sessions, démarrez une transaction

2) Dans la session 1 sélectionnez les code postaux des clients qui habitent Verviers

3) Dans la session 2, mettez ces codes postaux à jour

4) Recommencez le point 1

5) Dans la session 2, validez la transaction

6) Recommencez le point 1