Module 6: Vues, fonctions et procédures stockées

Une introduction

Les vues

Les vues: *protéger* votre modèle relationnel

Les vues correspondent à ce qu’on appelle « le niveau externe » qui reflète la partie visible de la base de données pour chaque utilisateur.

Les vues apparaissent comme des tables aux utilisateurs, des fenêtres sur la base de données.

Les vues: *protéger* votre modèle relationnel

Les vues permettent:

- d'avoir un meilleur contrôle sur l'accès aux données (**confidentialité**, sécurité et privilèges)
- de masquer la complexité d'une requête (ex: multiples jointures)
- de simplifier la formation de requêtes complexes

Les vues: *protéger* votre modèle relationnel

> En théorie, les utilisateurs ne devraient accéder aux informations *qu'en questionnant des vues*. Ces dernières masquent la structure réelle des tables (détails d'implémentation du schéma relationnel). En pratique, la plupart des applications se passent de ce concept en manipulant directement les tables (malheureusement).

En pratique, une vue n'est *qu'un nom de table donné à une requête d'extraction* (`SELECT`).

Il existe différents types de vues:

- **vues relationnelles (dématérialisées)**
- vues matérialisées
- vues semi-structurées

Nous ne verrons ici que les **vues relationnelles (dématérialisées) non modifiables**, c'est à dire accessibles en **lecture seule**.

Vues relationnelles dématérialisées

Une vue relationnelle est une *table virtuelle*. Elle n'occupe aucun espace disque (seulement chargée en mémoire, seule sa structure est stockée sur le disque).

On peut distinguer deux types de vues relationnelles:

- vue *simple* ou monotable
- vue *complexe* ou multitable

Création d'une vue simple (MySQL)

```
CREATE VIEW [nom base de données].nomTable
[ (alias1, alias2...)]
AS requete_select
[WITH CHECK OPTION}];
```

où:

- `alias` : nom de chaque colonne de la vue
- `requete_select` : la requête de sélection (`SELECT`)
- `WITCH CHECK OPTION` : garantit que la mise à jour de la vue (INSERT, UPDATE) respecte le prédicat de la définition des données (lors du CREATE)

> Pour en savoir plus, consulter la page sur la clause CREATE VIEW de la documentation officielle. Voir aussi cette page sur les restrictions sur les Vues en MySQL.

> Comme une vue se comporte comme une table, on peut créer *une vue d'une vue* ! Et ainsi de suite.

En pratique

> Voir le TD fourni, partie 1.

Création d'une vue complexe

Une vue complexe (ou multitable) est caractérisée par le fait qu'elle contient dans sa définition plusieurs tables (jointures) et une fonction appliquée à des regroupements ou des expressions.

> Les vues complexes vous encouragent donc à pousser le modèle relationnel jusqu'au bout et à créer autant de jointures que nécessaire. Vous pourrez ensuite masquer ces jointures dans des vues qui vous serviront de sources pour vos requêtes du côté applicatif. Il n'y a donc aucune excuse à ne pas faire des jointures et des jointures multiples !

Confidentialité

La **confidentialité est l'une des vocations premières des vues**. Grâce aux vues (et à une gestion des accès adéquate), les utilisateur·ices (développeurs·ses par exemple) d'une base de données ne peuvent pas accéder à des données confidentielles (projection dans la vue adéquate).

Fonctions et procédures stockées (*stored routines*)

Général

Le standard **SQL:1999** s'enrichit d'une syntaxe pour écrire des *procédures* ou *fonctions* dans un langage procédural.

> La norme SQL:1999 du langage procédural n'a pas été implémentée de manière rigoureuse par les différents éditeurs, car ils l'ont souvent implémenté avant l'apparition de la norme.

Dans un environnement client-serveur, **chaque instruction SQL donne lieu à l’envoi d’un message du client vers le serveur** suivi de la réponse du serveur vers le client.

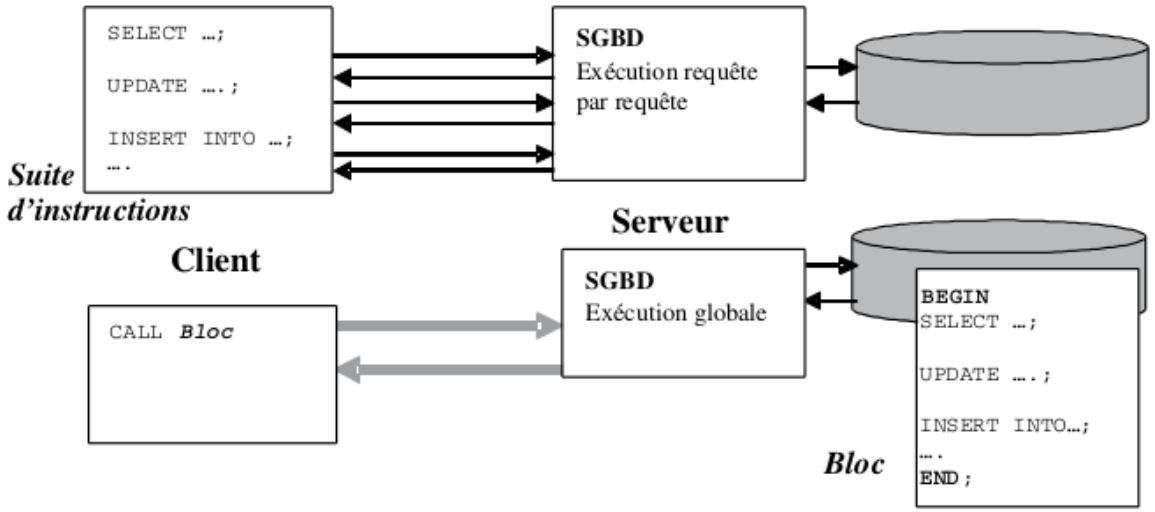
Il est donc préférable de travailler avec un sous-programme (qui sera stocké côté serveur) plutôt qu’avec une suite d’instructions SQL susceptibles d’encombrer le trafic réseau.

Usage

Une procédure stockée est un ensemble d'instructions SQL stocké (et compilé) directement sur le serveur MySQL. Une fois la procédure *stockée*, **le client n'a plus à transmettre un ensemble d'instructions au serveur, mais *une seule*** (celle qui déclenche l'ensemble des instructions).

En effet, un bloc donne lieu à un seul échange sur le réseau entre le client et le serveur. Les résultats intermédiaires sont traités côté serveur et **seul le résultat final est retourné au client**.

Figure 6-1 *Trafic sur le réseau d'instructions SQL*



Procédure stockée

Une *procédure stockée* est un programme SQL exécuté côté serveur par un appel d'un poste client (via `mysql` par exemple) ou par un appel interne depuis le serveur (une autre procédure stockée par exemple).

> Il est possible de *stocker* du code compilé dans tout type de langage (FORTRAN, COBOL, PASCAL, Perl, Java, C, Python, etc.) !

Avantages

- **modularité**, réutilisable
- peut être écrite dans n'importe quel langage supporté par le SGBD autre que SQL (C, Fotran, Java, Perl, etc.)*
- **placer de la logique métier dans le SGBD** et *hors du code applicatif*
- **portabilité** (hébergé par le SGBD)
- **performance en Input/Output** (moins de requêtes sur le réseau, temps d'attente, etc.)
- intégration naturelle avec les données des tables et les types du SGBD
- sécurité (*si* l'accès au SGBD est lui-même sécurisé)

> Pour en découvrir davantage, voir ici la page de la FAQ sur les procédures stockées dans MySQL.

Procédures stockées en MySQL

MySQL supporte les procédures stockées.

Tous les SGBD proposent un langage procédural pour écrire des fonctions ou procédures stockées.

MySQL ne supporte que des procédures stockées écrites en SQL. Impossible de faire référence à du code compilé écrit dans un autre langage.

> Pour en apprendre plus, voir la documentation.

Syntaxe de `CREATE PROCEDURE` (minimale)

```
delimiter//
CREATE PROCEDURE nomProcedure(IN arg INT)
BEGIN
 -- Déclarer des variables
 DECLARE foo CHAR(30) DEFAULT 'foo';
 SET foo  = 'bar';
 -- Corps de la procédure: requêtes SQL
 SELECT ...
 UPDATE ...
END//
delimiter;
CALL nomProcedure(1);
```

Exemple de procédure stockée

Reprendre la base de Module3/Exercice 1 (Client/Facture).

Création d'une procédure qui affiche les factures impayées depuis au moins X jours

```
DROP PROCEDURE listUnpaid;
-- on doit changer le délimiteur (point-virgule par défaut) de manière temporaire
-- pour écrire notre procédure
delimiter //
CREATE PROCEDURE listUnpaid(IN nbJours INT)
BEGIN
    SELECT DISTINCT c.prenom_client, c.nom_client, f.num_facture
    FROM Client c
    JOIN Facture f
    ON c.id_client = f.id_client
    WHERE
    NOT f.paye
    AND
    DATEDIFF(CURRENT_DATE(), f.date_emission) > nbJours;
END//
delimiter ;
-- lister les factures impayées depuis au moins 30 jours
CALL listUnpaid(30);
```

Exemple de procédure avec `SELECT INTO`

`SELECT INTO variableLocale` permet de stocker le résultat d'une requête dans une variable locale à la procédure.

Procédure qui retourne le déficit (le montant total des factures impayées), le nombre et le pourcentage de factures impayées, le nombre de factures payées.

```sql
DROP PROCEDURE compta;
delimiter //
CREATE PROCEDURE compta()
BEGIN
    -- Déclaration des variables locales
    DECLARE deficit FLOAT DEFAULT "0";
    DECLARE nbFacturesPayees INT DEFAULT 0;
    DECLARE nbFacturesImpayees INT DEFAULT 0;

    -- Stocke les résultats dans les variables locales
    SELECT SUM(montant_euro) INTO deficit FROM Facture WHERE NOT paye;
    SELECT COUNT(num_facture) INTO nbFacturesImpayees  FROM Facture WHERE NOT paye ;
    SELECT COUNT(num_facture) INTO nbFacturesPayees  FROM Facture WHERE paye ;

    -- Présentation du résultat
    SELECT
    deficit "Déficit (en EUROS)",
    nbFacturesImpayees / (nbFacturesImpayees + nbFacturesPayees) * 100 "%",
    nbFacturesImpayees  "Nb factures impayées",
    nbFacturesPayees  "Nb factures payées";

END//
-- on réinitialise le delimiter par défaut
delimiter ;
-- test
CALL compta;
```

Fonction (stockée)

Une fonction, comme une méthode, est un programme SQL, codé en SQL procédural ou dans un langage externe, qui accepte des paramètres en entrée et retourne une valeur ou une table (depuis la norme **SQL:2003**). Une fonction peut être appelée dans une requête ou dans l'instruction d'un traitement SQL.

Elle est déclarée avec l'instruction `CREATE FUNCTION`

> Attention une fonction en MySQL v8 ne peut pas renvoyer de table ! Seule une procédure peut.

```
-- définition d'une simple fonction
CREATE FUNCTION hello(s CHAR(20))
    RETURNS CHAR(50) DETERMINISTIC
    RETURN CONCAT('Hello, ',s,' !');
-- Appel de la fonction
SELECT hello('John');
```

Pour aller plus loin

- Les déclencheurs d'évènements (*triggers*). Voir ici la FAQ de la documentation officielle sur les triggers.
- Les transactions en MySQL
- Programmation avancée avec le langage procédure MySQL et les procédures/fonctions stockées: voir le chapitre 7 *Programmation avancée de l'ouvrage Apprendre SQL avec MySQL* de Christian Soutou, Eyrolles, 2006