Gestion des utilisateurs, des droits et des transactions en MariaDB

Gestion des utilisateurs et des droits

- Dans le cadre du cours, nous nous connectons à notre serveur MariaDB à l'aide d'un compte `root`, qui possède tous les droits sur le serveur.
- Dans un environnement de production, on évite à tout prix de travailler sur le serveur de bases de données avec un tel compte.
    - Pour assurer la sécurité du serveur et de nos données, on veut plutôt utiliser des comptes utilisateurs avec des droits beaucoup plus restreints.
    - Par exemple, chaque application utilisant notre serveur MariaDB pourrait avoir son propre compte utilisateur, avec uniquement les droits nécessaires au fonctionnement de l'application.
    - Dans un environnement de développement (tel que celui mis en place par XAMPP), il n'est pas rare que le compte `root` n'ait même pas de mot de passe. On ne doit absolument pas utiliser une telle configuration en production! Chaque compte utilisateur devrait avoir un mot de passe sécuritaire.

La base de données *mysql*

Les informations sur les utilisateurs et les droits sont conservés dans une base de données système appelée `mysql`. Cette base de données contient notamment les tables suivantes:

- `user` : Contient les informations sur les comptes utilisateur et leurs droits globaux.
- `db` : Contient les droits propres à des bases de données pour chaque utilisateur.
- `tables_priv`: Contient les droits propres à des tables ou des vues pour chaque utilisateur.
- `columns_priv`: Contient les droits propres à des champs de tables ou de vues pour chaque utilisateur.
- `procs_priv`: Contient les droits propres à des routines (procédures stockées et fonctions utilisateur) pour chaque utilisateur.

En faisant des requêtes `SELECT` sur ces tables, on peut voir les données qu'elles contiennent (à condition bien sûr d'avoir les droits le permettant). Les mots de passe des utilisateurs ne sont pas visibles directement, car ils sont hachés à l'aide de la fonction `PASSWORD`.

Pour gérer les utilisateurs et les droits, on ne modifie pas ces tables de la base de données `mysql` directement.

Gestion des utilisateurs

Créer un utilisateur

Pour créer un utilisateur, on utilise la commande `CREATE USER`. On doit lui donner un nom, et on peut lui attribuer un mot de passe (c'est d'ailleurs recommandé). Par exemple:

```
-- Création de l'utilisateur "harry", sans mot de passe
CREATE USER 'harry';

-- Création de l'utilisateur "hermione", avec le mot de passe "alohomora"
CREATE USER 'hermione' IDENTIFIED BY 'alohomora';
```

On peut aussi restreindre un utilisateur à un hôte ou un groupe d'hôtes en particulier. Par exemple:

```
/* Création de l'utilisateur "ron", qui peut seulement se connecter
   à partir de la machine qui héberge le serveur (localhost) */
CREATE USER 'ron'@'localhost' IDENTIFIED BY 'alohomora';

/* Création de l'utilisateur "ginny", qui peut seulement se connecter
   à partir d'une machine dont l'adresse IP commence par "192.168" */
CREATE USER 'ginny'@'192.168.%' IDENTIFIED BY 'alohomora';
```

Lorsqu'on ne spécifie pas d'hôte, c'est l'équivalent d'utiliser la valeur `%` pour celui-ci.

Finalement, on peut créer plusieurs configurations pour un même nom d'utilisateur:

```
/* Création de l'utilisateur "hagrid", autorisé
   à se connecter sans mot de passe
   à partir de la machine locale */
CREATE USER 'hagrid'@'localhost';

/* Permettre à hagrid de se connecter à partir des
autres hôtes à l'aide d'un mot de passe */
CREATE USER 'hagrid'@'%' IDENTIFIED BY 'gMU9UwUHT#7BNj9Dua';

/* Permettre à hagrid d'utiliser un autre mot de passe
à partir des adresses IP commençant par 192.168 */
CREATE USER 'hagrid'@'192.168.%' IDENTIFIED BY 'alohomora';
```

Supprimer un utilisateur

Pour supprimer un utilisateur, on utilise la commande `DROP USER`. Comme toujours avec les requêtes `DROP`, on peut ajouter la clause `IF EXISTS`. Comme pour les requêtes de création, la suppression d'un utilisateur se fait selon le nom d'hôte spécifié. Si celui-ci n'est pas indiqué, la valeur `%` est assumée.

```
DROP USER IF EXISTS 'harry'@'localhost';
```

Modifier un utilisateur

La commande `ALTER USER` permet de modifier un utilisateur. Elle est surtout utile pour changer son mot de passe. Par exemple

```
ALTER USER 'hagrid'@'%' IDENTIFIED BY 'Yq@^xxJe%%R2xSHCKLnh';
```

On peut utiliser la fonction `CURRENT_USER` pour changer le mot de passe de l'utilisateur courant.

```
ALTER USER CURRENT_USER() IDENTIFIED BY 'monSuperNouveauMotDePasse';
```

Gestion des droits

- Par défaut, un utilisateur a uniquement le droit `USAGE`, qui permet de se connecter au serveur.
- Pour qu'un utilisateur puisse effectuer des opérations sur le serveur, il faut lui attribuer d'autres droits.

Attribuer un droit

La commande `GRANT` permet d'attribuer un droit à un utilisateur. Si l'utilisateur n'existe pas déjà, il sera créé.

Pour donner tous les droits à un utilisateur sur un serveur, on utilise:

```
GRANT ALL PRIVILEGES ON *.* TO 'dumbledore'@'localhost';
```

ATTENTION: un tel niveau de privillège devrait être réservé aux administrateurs!

Si on veut qu'en plus, l'utilisateur puisse attribuer n'importe quel droit à d'autres utilisateurs, on ajoute l'option `WITH GRANT OPTION`.

```
GRANT ALL PRIVILEGES ON *.* TO 'dumbledore'@'localhost' WITH GRANT OPTION;
```

On peut aussi spécifier la base de données sur laquelle on veut donner tous les droits à un utilisateur. Par exemple:

```
GRANT ALL PRIVILEGES ON librairie.* TO `harry`@`localhost`;
```

On peut même donner des droits uniquement sur une table spécifique:

```
GRANT ALL PRIVILEGES ON librairie.livre TO `hermione`@`localhost`;
```

Au lieu de donner tous les droits à un utilisateur, on peut aussi vouloir lui donner des droits spécifiques. La liste complète des droits possibles est disponible ici.

Voici deux exemples:

```
-- Donner le droit de créer des tables sur la base de données `commerce`
GRANT CREATE ON commerce.* TO 'ron'@'%';

-- Donner le droit d'exécuter des requêtes `SELECT` et `INSERT` sur la table
`livre`
GRANT SELECT, INSERT ON librairie.livre TO `ron`@'%';
```

Retirer un droit

On peut retirer des droits à un utilisateur à l'aide de la commande `REVOKE ... FROM`. On peut supprimer tous les droits d'un utilisateur (`ALL PRIVILLEGES`), ou des droits spécifiques. Par exemple:

```
REVOKE ALL PRIVILEGES ON *.* FROM 'dumbledore'@'localhost';
REVOKE SELECT, CREATE ON librairie.* FROM 'harry'@'%';
```

Voir les droits

La commande `SHOW GRANTS` permet de visualiser les droits d'un utilisateur.

```
SHOW GRANTS FOR 'ron'@'localhost';
```

Les rôles

Les rôles permettent d'attribuer facilement les mêmes droits à plusieurs utilisateurs. Pour ce faire, on crée un rôle avec `CREATE ROLE`, puis on attribue des droits à ce rôle avec `GRANT`, avant d'attribuer le rôle à des utilisateurs. Par exemple:

```
CREATE ROLE 'superadmin';
GRANT ALL PRIVILEGES ON *.* to 'superadmin';
GRANT 'superadmin' TO 'dumbledore'@'localhost';
```

Sécuriser les vues et les procédures stockées

Même si on donne à un utilisateur le droit d'accéder aux données d'une vue en particulier, celui-ci ne pourra pas le faire si la vue utilise des données provnnant d'une table à laquelle il n'a pas le droit d'accéder. Pour résoudre ce problème, on utilise la clause `SQL SECURITY` lors de la création de la vue. Celle-ci a deux valeurs possibles: `INVOKER` pour que la vue s'exécute avec les droits de l'utilisateur qui invoque la vue, ou `DEFINER` pour qu'elle s'exécute avec les droits de l'utilisateur qui a créé la vue. Par exemple:

```sql
CREATE OR REPLACE SQL SECURITY DEFINER VIEW vue_livres AS
    SELECT  l.isbn,
            l.titre,
            CONCAT_WS(' ', a.prenom, a.nom) AS auteur,
            e.nom AS editeur,
            l.date_parution,
            l.description,
            l.code_langue
        FROM livre l
        JOIN auteur_livre al
            ON al.isbn_livre = l.isbn
        JOIN auteur a
            ON a.id = al.id_auteur
        JOIN editeur e
            ON e.id = l.id_editeur;
```

Le même principe s'applique aux procédures stockées.

Les transactions

Une transaction sur un SGBD permet d'exécuter une séquence de requêtes SQL de façon atomique, c'est-à-dire que si pour une raison quelconque (ex: erreur ou défaillance) une des requêtes n'arrive pas à être exécutée, toutes les requêtes de la transaction seront annulées. En bon français québécois, quand on exécute des requêtes dans une transaction, c'est « toute ou pantoute » !

Les commandes de base pour utiliser une transaction sont `START TRANSACTION`, pour démarrer la transaction et `COMMIT` pour appliquer ses effets. On peut aussi utiliser `ROLLBACK` pour annuler les requêtes de la transaction courante. Il est à noter que seules les requêtes de type DML, et non celles de type DDL, peuvent être annulées.

Références

- Documentation officielle de MariaDB:
    - Sur la gestion des utilisateurs et des droits
    - Sur les transactions