Les requêtes de définition de données en MariaDB

Introduction

- MariaDB utilise le langage SQL, comme pratiquement tous les systèmes de gestion de bases de données relationnelles (SGBDR).
- Le langage SQL est composé d'instructions dont chacune appartient à l'une des catégories suivantes:
    - Langage de définition de données (DDL pour *Data Definition Language*)
    - Langage de manipulation de données (DML pour *Data Manipulation Language*)
    - Langage de contrôle de données (DCL pour *Data Control Language*)
    - Langage de contrôle des transactions (TCL pour *Transaction Control Language*)
- Les instructions couvertes dans ce document sont celles du DDL.
- Bien que les bases du langage SQL sont les mêmes d'un SGBDR à l'autre, il existe des petites différences. On parle donc de différents « dialectes » du langage SQL.
- Le dialecte que nous allons utiliser dans le cours est bien sûr celui de MySQL/MariaDB.

Syntaxe générale

- Une requête SQL commence toujours par un verbe et se termine toujours par un point-virgule (`;`).
- Une requête de type DDL est formée de mots-clés et d'identifiants.
- Un mot-clé est un mot qui fait partie du langage SQL, tandis qu'un identifiant est un nom donné par l'utilisateur à un élément de la base de données, par exemple: un nom de table, un nom d'attribut, etc.
- Les mots-clés ne sont pas sensibles à la casse, par contre par convention on les écrit en majuscules.
- Avec MariaDB, la sensibilité à la casse des identifiants dépend par défaut du système d'exploitation qui héberge la base de données (pas sensible à la casse sur Windows, mais sensible à la casse sur Linux et macOS).
- Dans le cours, on se donnera pour convention d'écrire nos identifiants tout en minuscules et en *snake_case*.

Voici un exemple de requête SQL:

```
CREATE DATABASE location_vehicules;
```

Dans cet exemple, `CREATE` et `DATABASE` sont des mots-clés, et `location_vehicules` est un identifiant.

On peut aussi écrire n'importe quelle requête sur plusieurs lignes, par exemple:

```

```

```
CREATE
    DATABASE location_vehicules;
```

Commentaires

On peut ajouter des commentaires en SQL. Un commentaire sert à fournir des explications supplémentaires, et n'est pas pris en compte lorsqu'on exécute le script.

```
-- Commentaire sur une seule ligne

/*
Pour écrire un commentaire sur plusieurs lignes, on place
une barre oblique suivie d'un astérisque au début du commentaire,
et un astérisque suivi d'une barre oblique à la fin du commentaire.

Cette même syntaxe est utilisée dans de nombreux langages, dont PHP.
*/
```

Les commentaires peuvent aussi être utilisés lorsqu'on veut éviter d'exécuter certaines lignes d'un script SQL, mais sans les effacer.

Créer une base de données

La syntaxe pour créer une base de données est `CREATE DATABASE <nom de la BD>;`.

Par exemple, pour créer une base de données nommée "librairie":

```
CREATE DATABASE librairie;
```

On peut aussi ajouter l'expression `IF NOT EXISTS` devant le nom de la base de données afin de la créer seulement si elle n'existe pas déjà. Par exemple:

```
CREATE DATABASE IF NOT EXISTS librairie;
```

Avant de créer la base de données, il faut s'assurer qu'on travaille sur le bon serveur.

Supprimer une base de données

La syntaxe pour supprimer une base de données est `DROP DATABASE <nom de la BD>;`.

Par exemple, pour supprimer la base de données nommée `librairie`:

```
DROP DATABASE librairie;
```

On peut aussi ajouter l'expression `IF EXISTS` devant le nom de la base de données afin de la supprimer seulement si elle existe. Par exemple:

```
DROP DATABASE IF EXISTS librairie;
```

Utiliser une base de données

Avant d'exécuter des requêtes SQL, il est important de s'assurer qu'on travaille sur la bonne base de données. On utilise pour cela la commande `USE`:

```
USE librairie;
```

Créer des tables

La commande `CREATE TABLE` permet de créer une table. Sa syntaxe générale est la suivante:

```
CREATE TABLE <nom de la table> (
    <nom du premier champ> <type de champ> <options>,
    <nom du deuxième champ> <type de champ> <options>,
    <nom du troisième champ> <type de champ> <options>
);
```

Voici par exemple comment ajouter une table `editeur` à notre base de données `librairie`:

```
USE librairie; -- On place cette ligne une seule fois, après la création de la BD

CREATE TABLE editeur (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL
);
```

Dans cet exemple, on crée une table nommée `editeur` avec deux colonnes:

- `id`, de type `INT`, c'est-à-dire un nombre entier dont la valeur est entre -2147483648 et 2147483647 (environ 2,1 milliards).
- `nom`, de type `VARCHAR` (chaîne de caractères à longueur variable) dont la longueur maximale est de 50 caractères. Ce champ n'autorise pas les valeurs nulles (`NOT NULL`).

Par ailleurs, on indique que le champ `id` obtiendra une valeur générée automatiquement qui s'incrémentera chaque fois qu'on ajoutera une entrée à la table (`AUTO_INCREMENT`), et qu'il s'agit de la clé primaire de la table (`PRIMARY KEY`).

On aurait aussi pu ajouter `IF NOT EXISTS` à la commande (`CREATE TABLE IF NOT EXISTS editeur`).

Créons maintenant une table `livre` contenant une clé étrangère faisant référence à un ID d'éditeur.

```
CREATE TABLE IF NOT EXISTS livre (
    isbn CHAR(13) PRIMARY KEY,
    titre VARCHAR(50) NOT NULL,
    id_editeur INT,
    annee_publication CHAR(4) NOT NULL,
    description TEXT,
    CONSTRAINT fk_livre_editeur
        FOREIGN KEY (id_editeur) REFERENCES editeur(id)
        ON DELETE CASCADE
        ON UPDATE RESTRICT
);
```

Dans cette table, on a:

- Un champ `isbn` de type `CHAR` (chaîne de caractères à longueur fixe) d'une longueur de 13 caractères. On utilise ce champ comme clé primaire.
- Un champ `titre` de type `VARCHAR` d'une longueur maximale de 20 caractères, qui ne peut pas être nul.
- Un champ `id_editeur` de type `INT`, qui peut être nul.
- Un champ `annee_publication` de type `CHAR` d'une longeur de 4, qui ne peut pas être nul.
- Un champ `description` de type `TEXT`, un type optimisé pour les longues chaînes de caractères (jusqu'à 65535 caractères). Ce champ accepte les valeurs nulles.
- Une clé étrangère de l'attribut `id_editeur` vers l'attribut `id` de la table `editeur`.

La clause `ON DELETE CASCADE` a pour effet de supprimer automatiquement les livres d'un éditeur lorsqu'on supprime cet éditeur, tandis que la clause `ON UPDATE RESTRICT` interdit de modifier l'ID d'un éditeur si des livres y sont associés. Les autres options possibles pour ces deux clauses sont:

- `NO ACTION`: synonyme de `RESTRICT`
- `SET NULL`: vide le champ.
- `SET DEFAULT`: réinitialise le champ à sa valeur par défaut (défini par l'option `DEFAULT <valeur par défaut>` dans la création du champ).

L'option par défaut (si on n'inclut pas la clause) est `RESTRICT` pour les deux clauses.

Une syntaxe alternative pour définir une clé étrangère est la suivante:

```
CREATE TABLE IF NOT EXISTS livre (
    isbn CHAR(13) PRIMARY KEY,
    titre VARCHAR(50) NOT NULL,
    id_editeur INT,
    annee_publication CHAR(4) NOT NULL,
    description TEXT,
    FOREIGN KEY (id_editeur) REFERENCES editeur(id)
```

```
        ON DELETE CASCADE
        ON UPDATE RESTRICT
);
```

Cette syntaxe a pour inconvénient d'utiliser un nom généré par le système pour la contrainte de clé étrangère. Cela est moins pratique si on veut par la suite supprimer la clé.

Créons maintenant une table `pays`, puis une table `auteur`:

```
CREATE TABLE IF NOT EXISTS pays (
    code CHAR(2) PRIMARY KEY,
    nom VARCHAR(50) NOT NULL
);

CREATE TABLE IF NOT EXISTS auteur (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    code_pays CHAR(2) NOT NULL,
    FOREIGN KEY (code_pays) REFERENCES pays(code)
        ON DELETE RESTRICT
        ON UPDATE RESTRICT
);
```

Puis, une table d'association entre la table `auteur` et la table `livre`. Nous appellerons cette table: `auteur_livre`.

```
CREATE TABLE IF NOT EXISTS auteur_livre (
    id_auteur INT NOT NULL,
    isbn_livre CHAR(13) NOT NULL,
    PRIMARY KEY (id_auteur, isbn_livre),
    FOREIGN KEY (id_auteur) REFERENCES auteur(id)
        ON DELETE RESTRICT
        ON UPDATE RESTRICT,
    FOREIGN KEY (isbn_livre) REFERENCES livre(isbn)
        ON DELETE RESTRICT
        ON UPDATE RESTRICT
);
```

Remarquons que cette table utilise une clé primaire composite, c'est-à-dire formée de plusieurs champs (les champs `id_auteur` et `isbn_livre`).

Types de champs

Les types de champs qu'on rencontre les plus fréquemment sont les suivants:

- Types numériques
    - Nombres entiers

        - `TINYINT` (entre -128 et 127)
        - `SMALLINT` (entre -32768 et 32767)
        - `MEDIUMINT` (entre -8388608 et 8388607)
        - `INT` (entre -2147483648 et 2147483647)
        - `BIGINT` (entre -9223372036854775808 et 9223372036854775807)
        - Versions non signées (ex: `INT UNSIGNED` entre 0 et 4294967295)
    - Nombres à virgule
        - `FLOAT`
        - `DOUBLE`
        - `DECIMAL` (pour des valeurs exactes avec un nombre fixe de chiffres après la virgule, recommandé pour les montants monétaires)
    - Chaînes de caractères
        - `CHAR`
        - `VARCHAR`
        - `TEXT`
    - Booléens
        - `BOOLEAN` (valeurs possibles: `TRUE` ou `FALSE`, stocké sous forme de `TINYINT`)
    - Dates et heures
        - `DATE`
        - `TIME`
        - `DATETIME`
        - `TIMESTAMP`

Pour toutes les informations sur les types de champs, se référer à cette page de la documentation officielle de MariaDB.

Modifier une table

La commande `ALTER TABLE` permet de modifier une table. Voici plusieurs exemples de modification de la table `livre`.

```
-- Ajouter une colonne à une table
ALTER TABLE livre
    ADD COLUMN langue VARCHAR(20);

-- Retirer une clé étrangère d'une table (cela ne supprime pas le champ)
ALTER TABLE livre
    DROP FOREIGN KEY fk_livre_editeur;

/*
Ajouter une clé étrangère à une table.
Pratique si la table référée n'existait pas encore à la création de la table qui y
fait référence.
*/
ALTER TABLE livre
    ADD FOREIGN KEY (id_editeur) REFERENCES editeur(id);

-- Modifier un champ
```

```
ALTER TABLE livre
    MODIFY COLUMN langue CHAR(2) NOT NULL DEFAULT 'FR';

-- Supprimer un champ
ALTER TABLE livre
    DROP COLUMN langue;
```

Pour voir toutes les possibilités de la commande `ALTER TABLE`, se référer à cette page de la documentation officielle de MariaDB.

Supprimer une table

Pour supprimer une table, on utilise la commande `DROP TABLE`.

Par exemple:

```
DROP TABLE auteur_livre;
```

Ou encore:

```
DROP TABLE IF EXISTS auteur_livre;
```

On ne peut pas supprimer une table vers laquelle d'autres tables contiennent des clés étrangères. Il faut donc d'abord supprimer ces tables, ou supprimer les clés étrangères.

Références

- Documentation officielle du DDL de MariaDB
- Tutoriel sur MySQL de W3Schools