Créer, gérer et utiliser une base de données avec MySQL

Table des matières

- **Vidéos**: *https://www.youtube.com/playlist?list=PLFtlZvb4qwees1u9G58lu0Vq_Dzj4zX5N*
- **Documentation**: *https://sql.sh/*

Le SGBDR MySQL

Pour illustrer ce cours nous allons utiliser le SGBDR MySQL. Il en existe d'autres comme PostgreSQL ou MariaDB.

MySQL est un système de gestion de bases de données relationnelles SQL open source développé et supporté par Oracle.

Une base de données n'est qu'une collection structurée de données qui est organisée pour en faciliter l'utilisation et la récupération. Par exemple, pour un site WordPress, ces « données » peuvent être des éléments comme le texte de vos articles de blog, des informations pour tous les utilisateurs enregistrés sur votre site, des données chargées automatiquement, des configurations de paramètres importants, etc.

MySQL est un système populaire qui peut stocker et gérer ces données pour vous, et c'est une solution de base de données particulièrement populaire pour les sites développés en php.

Qu'est-ce qu'une base de données relationnelle ?

Lorsqu'il s'agit de stocker des données dans une base de données, il existe différentes approches que vous pouvez utiliser. MySQL opte pour une approche appelée base de données relationnelle.

Avec une base de données relationnelle, vos données sont divisées en plusieurs zones de stockage séparées – appelées tables – (les entités des MCD) plutôt que de tout regrouper dans une seule grande unité de stockage.

Par exemple, disons que vous voulez stocker deux types d'informations :

- Les clients – leur nom, leur adresse, leurs coordonnées, etc.
- Les commandes – par exemple, quels produits ont été achetés, le prix, qui a passé la commande, etc.

Si vous essayiez de regrouper toutes ces données dans un seul grand pot, vous aurez quelques problèmes comme :

- Données différentes – les données que vous devez collecter pour une commande sont différentes de celles d'un client.
- Dupliquer les données – chaque client a un nom, et chaque commande a aussi le nom d'un client. Le traitement de ces données en double devient désordonné.
- Pas d'organisation – comment pouvez-vous relier de façon fiable les informations relatives aux commandes aux informations sur les clients ?

Pour résoudre ces problèmes, une base de données relationnelle utilisera une table séparée pour les clients et une autre table séparée pour les commandes. Cependant, vous voudrez probablement aussi pouvoir dire « montrez-moi toutes les commandes réalisées Jean-Michel Dupont ». C'est là qu'intervient la partie relationnelle.

En utilisant ce qu'on appelle une « clé » (les id dans les MCD), vous pouvez lier les données de ces deux tables entre elles afin de pouvoir les manipuler et les combiner dans différentes tables si nécessaire. Il est important de noter qu'une clé n'est pas le nom du client. Au lieu de cela, vous utiliserez quelque chose de 100% unique, comme un numéro d'identification numérique.

Le SQL

Structured **Q**uery **L**anguage

Le SQL, "Langage de requête structurée" en français est tout simplement comme son nom l'indique un langage permettant de créer, gérer et utiliser des bases de données.

Les principales instructions du SQL

Le SELECT

L'instruction SELECT va interroger la base de données pour fournir le contenu de certaines tables en fonction de critères. Dans l'instruction SELECT, on peut :

- Choisir les champs à afficher
- Choisir les tables d'où proviennent les données
- Faire des tris pour afficher les résultats dans un ordre choisi
- Regrouper certaines valeurs en fonction de critères
- Réaliser des calculs (des sommes par exemple), des tris pour affiner les résultats
- etc ...

C'est certainement l'instruction SQL la plus utilisée puisque c'est celle qui nous permet de sélectionner l'information à afficher.

Cette instruction se présente sous cette forme :

SELECT nom _colonnes **FROM** nom_table

Exemple : SELECT `ville_nom` FROM `villes_france_free`

Nous pouvons bien évidemment choisir d'afficher plusieurs colonnes. Pour cela, il suffit de séparer les deux noms de colonnes par une virgule.

Exemple : SELECT `ville_nom`, `ville_departement` FROM `villes_france_free`;

Le WHERE

L'instruction WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition que l'on va définir. On obtient ainsi uniquement les informations que l'on souhaite sélectionner. Cet opérateur se présente sous cette forme :

SELECT nom_colonnes **FROM** nom_table **WHERE** condition

Exemple : SELECT `ville_nom` FROM `villes_france_free` WHERE `ville_departement` = 44;

Les opérateurs de comparaisons

Afin de pouvoir trier les données, il existe plusieurs opérateurs de comparaison permettant de compléter la l'instruction WHERE :

Opérateur	Signification
=	Égal à
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à
!=	Différent de
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

Les opérateurs AND et OR

Les opérateurs AND et OR sont des opérateurs logiques qui peuvent être utilisés dans un WHERE afin de rendre le tri encore plus précis. Cet opérateur se présente sous cette forme :

SELECT nom_colonnes **FROM** nom_table **WHERE** condition1 **AND** condition2
ou
SELECT nom_colonnes **FROM** nom_table **WHERE** condition1 **OR** condition2

Exemple : SELECT `ville_nom` FROM `villes_france_free` WHERE `ville_departement` = 44 AND `ville_densite_2010` >= 100 ;

Il est également possible de combiner les opérateurs.

Exemple : SELECT `ville_nom` FROM `villes_france_free`
WHERE (`ville_departement` = 44 AND `ville_densite_2010` >= 100)
OR (`ville_departement` = 50 AND `ville_densite_2010` >= 200) ;

L'opérateur BETWEEN

L'opérateur BETWEEN est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant WHERE. Cet opérateur se présente sous cette forme :

SELECT * **FROM** table **WHERE** nom_colonne **BETWEEN** 'valeur1' **AND** 'valeur2'

L'opérateur LIKE

Toujours utilisé dans l'instruction WHERE, ce mot clé permet de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Cet opérateur se présente sous cette forme :

SELECT * **FROM** table **WHERE** colonne **LIKE** modèle

Voici les différents modèles en fonction de ce que l'on recherche :

- LIKE '%a' : le caractère "%" est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaines de caractère qui se termine par un "a".
- LIKE 'a%' : ce modèle permet de rechercher toutes les lignes de "colonne" qui commence par un "a".
- LIKE '%a%' : ce modèle est utilisé pour rechercher tous les enregistrements qui utilisent le caractère "a".
- LIKE 'pa%on' : ce modèle permet de rechercher les chaines qui commence par "pa" et qui se terminent par "on", comme "pantalon" ou "pardon".
- LIKE 'a_c' : peu utilisé, le caractère "_" (underscore) peut être remplacé par n'importe quel caractère, mais un seul caractère uniquement (alors que le symbole pourcentage "%" peut être remplacé par un nombre incalculable de caractères . Ainsi, ce modèle permet de retourner les lignes "aac", "abc" ou même "azc".

Les opérateurs de IS NULL et IS NOT NULL

L'opérateur IS permet de filtrer les résultats qui contiennent la valeur NULL. Cet opérateur est indispensable car la valeur NULL est une valeur inconnue et ne peut par conséquent pas être filtrée par les opérateurs de comparaison (cf. égal, inférieur, supérieur ou différent). Cet opérateur se présente sous cette forme :

SELECT * **FROM** `table` **WHERE** nom_colonne **IS NULL**

ou

SELECT * **FROM** `table` **WHERE** nom_colonne **IS NOT NULL**

Le GROUP BY

L'instruction GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat. Sur une table qui contient toutes les villes d'un pays, il est possible de regrouper les villes par département. Cette instruction se présente sous cette forme :

SELECT colonne1, fonction(colonne2) **FROM** table **GROUP BY** colonne1

Exemple : SELECT `ville_nom`, `ville_departement` FROM `villes_france_free` GROUP BY `ville_departement`;

Le ORDER BY

Comme son nom l'indique, le ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant. Cette instruction se présente sous cette forme :

SELECT colonne1, colonne2 **FROM** table **ORDER BY** colonne1 **ASC**

ou

SELECT colonne1, colonne2 **FROM** table **ORDER BY** colonne1 **DESC**

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne. De ce fait nous ne sommes pas obligés d'utiliser le suffixe "ASC" sachant que les résultats sont toujours classés par ordre ascendant par défaut.

Le HAVING

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX(). L'utilisation de HAVING s'utilise de la manière suivante :

SELECT colonne1, fonction(colonne2) **FROM** nom_table **GROUP BY** colonne1 **HAVING** fonction(colonne2) opérateur valeur

Les fonctions SQL

Liste des fonctions d'agrégation statistiques

Les fonctions d'agrégation sont des fonctions idéales pour effectuer quelques statistiques de bases sur des tables. Les principales fonctions sont les suivantes :

- **AVG()** pour calculer la moyenne sur un ensemble d'enregistrement

  Exemple : SELECT AVG(`ville_population_2012`) FROM `villes_france_free`
- **COUNT()** pour compter le nombre d'enregistrement sur une table ou une colonne distincte.

  Exemple : SELECT COUNT(*) FROM `villes_france_free` WHERE `ville_departement`=44
- **MAX()** pour récupérer la valeur maximum d'une colonne sur un ensemble de ligne. Cela s'applique à la fois pour des données numériques ou alphanumérique
- **MIN()** pour récupérer la valeur minimum de la même manière que MAX()
- **SUM()** pour calculer la somme sur un ensemble d'enregistrement

  Exemple : SELECT SUM(`ville_population_2012`) FROM `villes_france_free`

Liste des fonctions de chaînes de caractères

Il existe un grand nombre de fonctions qui concernent les chaînes de caractères (voir le site *sql.sh*). Voici les plus utilisées :

- LENGTH() retourner la longueur d'une chaîne
- REPLACE() remplacer des caractères par d'autres caractères
- RTRIM() supprimer les caractères vides en fin d'une chaîne de caractère
- TRIM() supprime les caractères vides en début et fin de chaîne
- UPPER() tout retourner en majuscule
- ...

Il existe également des fonctions concernant les dates ou le chiffrement et je vous invite à jeter un œil sur la documentation si vous désirez en savoir plus.

La clause LIMIT

La clause LIMIT est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'on souhaite obtenir. Cette clause est souvent associée à un OFFSET, c'est-à dire effectuer un décalage sur le jeu de résultat. Ces 2 clauses permettent par exemple d'effectuer des systèmes de pagination (exemple : récupérer les 10 articles de la page 4).

La clause LIMIT s'utilise de la manière suivante :

```
SELECT * FROM table LIMIT 10
```

On peut ainsi, par exemple chercher la ville la plus peuplée de France :

```
SELECT `ville_nom`, `ville_population_2012`
FROM `villes_france_free`
ORDER BY `ville_population_2012` DESC LIMIT 1
```

Les alias AS

Permet de renommer le nom d'une colonne dans les résultats d'une requête SQL. C'est pratique pour avoir un nom facilement identifiable dans une application qui doit ensuite exploiter les résultats d'une recherche.

Exemple : SELECT COUNT(*) AS Nb_Ville FROM `villes_france_free` WHERE `ville_departement`=44

Les jointures SQL

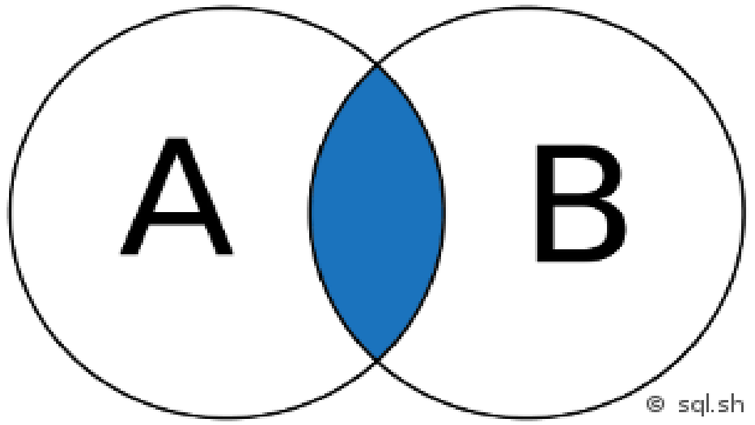
Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

INNER JOIN

La jointure INNER JOIN s'utilise de la manière suivante :

SELECT * **FROM** A **INNER JOIN** B **ON** A.key = B.key

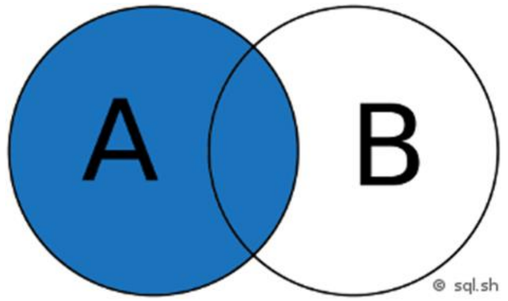
Exemple : SELECT `ville_nom`,`departement_nom`
FROM `villes_france_free`
INNER JOIN departement ON ville_departement = departement_code



LEFT JOIN

La jointure LEFT JOIN s'utilise de la manière suivante :

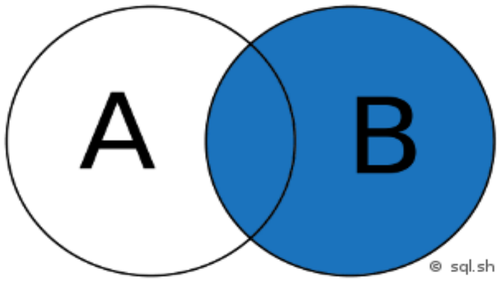
SELECT * **FROM** A **LEFT JOIN** B **ON** A.id = B.id



RIGHT JOIN

La jointure RIGHT JOIN s'utilise de la manière suivante :

SELECT * **FROM** A **RIGHT JOIN** B **ON** A.id = B.id



L'instruction CREATE DATABASE

La création d'une base de données en SQL est possible en ligne de commande. Même si les systèmes de gestion de base de données (SGBD) sont souvent utilisés pour créer une base, il convient de connaître la commande à utiliser, qui est très simple. Voici sa syntaxe:

CREATE DATABASE ma_base_de_donnee

ou

CREATE DATABASE IF NOT EXISTS ma_base_de_donnee

L'instruction CREATE TABLE

La commande CREATE TABLE permet de créer une table en SQL. Un tableau est une entité qui est contenu dans une base de données pour stocker des données ordonnées dans des colonnes. La création d'une table sert à définir les colonnes et le type de données qui seront contenus dans chacun des colonne (entier, chaîne de caractères, date, valeur binaire ...).

Voici sa syntaxe:

```
CREATE TABLE IF NOT EXISTS ma_base_de_donnee.nom_de_la_table(
    colonne1 type_donnees,
    colonne2 type_donnees,
    colonne3 type_donnees,
    colonne4 type_donnees
)
```

L'instruction INSERT INTO

L'ajout de données dans une table s'effectue grâce à l'instruction INSERT INTO. Cette commande permet d'ajouter une seule ligne à une table ou plusieurs lignes d'un coup. Voici sa syntaxe:

```
INSERT INTO table VALUES ('valeur 1', 'valeur 2', ...)
```

Ou pour le cas où vous souhaitez ajouter plusieurs lignes à votre table :

```
INSERT INTO table VALUES ('valeur 1', 'valeur 2', ...),
                         ('valeur 1', 'valeur 2', ...),
                         ('valeur 1', 'valeur 2', ...),
                         ...
```

L'instruction UPDATE

La commande UPDATE permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec WHERE pour spécifier sur quelles lignes doivent porter la ou les modifications. Elle s'utilise sous cette forme :

```
UPDATE table
SET nom_colonne_1 = 'nouvelle valeur'
WHERE condition
```

Exemple :
```
UPDATE villes_france_free
    SET ville_nom = 'Quilly en Campagne'
    WHERE ville_nom = 'Quilly'
```

L'instruction DELETE

La commande DELETE en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associée à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées. Elle s'utilise sous cette forme:

```
DELETE FROM table
WHERE condition
```

Les sous-requêtes

Dans le langage SQL une sous-requête (aussi appelé “requête imbriquée” ou “requête en cascade”) consiste à exécuter une requête à l’intérieur d’une autre requête. Une requête imbriquée est souvent utilisée au sein d’une clause WHERE ou de HAVING pour remplacer une ou plusieurs constantes. Elles s’utilisent sous cette forme :

```
SELECT * FROM `table` WHERE `nom_colonne` = (
    SELECT `valeur`
    FROM `table2`
    LIMIT 1
)
```