



Documentation MySQL

▼ Version	V1
≡ Type	Technique
📅 Date de création	@12 février 2024 13:32
📅 Dernière modification	@7 mars 2024 13:56

🚀 Commandes SQL



CRUD

Create - Read - Update - Delete

Dans la présentation suivante nous utiliserons une table comme celle-ci :

id	nom	prenom	email
1	FOO	John	sf@gmail.com
2	BAR	Mike	m_bar_123@gmail.com
3	TREE	Sarah	tree.sarah@gmail.com

▼ 🔍 Read

```
SELECT email FROM utilisateur; # Selectionne les email de la table utilisateur
SELECT nom, prenom FROM utilisateur; # Selectionne les noms et prenom de la table utilisateur
SELECT * FROM utilisateur; # Selectionne tous les enregistrements de la table utilisateur
```

Exemple générique

```
SELECT une_colonne FROM une_table; # Pour sélectionner une colonne
SELECT une_colonne, une_seconde_colonne FROM une_table; # Pour sélectionner plusieurs colonnes
SELECT * FROM une_table; # Pour sélectionner toutes les colonnes
```

i On peut affiner nos recherches, les conditionner, agréger nos données, etc

▼ AS - alias

En SQL, il est possible d'utiliser des alias. Il s'agit de **renommer temporairement une colonne ou une table dans une requête**. Cette fonctionnalité est très pratique pour faciliter la lecture des résultats d'une requête SQL. Au niveau des tables, **l'intérêt d'utiliser des alias se vérifie lorsqu'il y a des jointures**. Cela permet notamment d'obtenir des noms plus courts pour les tables.

```
SELECT une_colonne AS ce_que_vous_voulez FROM une_table; # Utiliser un alias sur un  
SELECT * FROM une_table AS ce_que_vous_voulez; # Utiliser un alias sur une table
```

▼ LIMIT et OFFSET - limite de résultat

En SQL, la commande `LIMIT` permet de **spécifier le nombre maximum de résultats** que l'on souhaite obtenir, tandis que la commande `OFFSET` permet d'**effectuer un décalage sur l'ensemble des résultats**.

```
SELECT * FROM une_table LIMIT une_limite; # Définir un nombre maximum de résultats  
SELECT une_colonne FROM une_table OFFSET un_offset; # Définir un décalage dans les
```

▼ Fonctions d'agrégation

En SQL, les fonctions d'agrégation permettent de **réaliser des opérations arithmétiques et statistiques** au sein d'une requête.

- `COUNT()` pour compter le nombre d'enregistrements d'une table ou d'une colonne distincte ;
- `AVG()` pour calculer la moyenne sur un ensemble d'enregistrements ;
- `SUM()` pour calculer la somme sur un ensemble d'enregistrements ;
- `MAX()` pour récupérer la valeur maximum d'une colonne sur un ensemble d'enregistrements ;
- `MIN()` pour récupérer la valeur minimum de la même manière que la fonction `MAX()`.

Utiliser une fonction.

```
SELECT votre_fonction(une_colonne) FROM une_table;
```

COUNT

En SQL, la fonction d'agrégation `COUNT()` permet de **compter le nombre d'enregistrements** dans une table, les enregistrements qui possèdent la valeur **NULL ne seront pas comptabilisés**.

```
SELECT COUNT(une_colonne) FROM une_table;
```

AVG

En SQL, la fonction d'agrégation `AVG()` permet de **calculer une valeur moyenne sur un ensemble d'enregistrements** de type numérique et **qui ne possèdent pas la valeur NULL**.

```
SELECT AVG(une_colonne) FROM une_table;
```

SUM

En SQL, la fonction d'agrégation `SUM()` permet de **calculer la somme sur un ensemble d'enregistrements** de type numérique et **qui ne possèdent pas la valeur NULL**.

```
SELECT SUM(une_colonne) FROM une_table;
```

MIN et MAX

En SQL, les fonctions d'agrégation `MAX()` et `MIN()` permettent de **sélectionner la valeur respectivement maximale et minimale dans un ensemble d'enregistrements. Les valeurs NULL sont écartées.**

```
SELECT MAX(une_colonne) FROM une_table;
SELECT MIN(une_colonne) FROM une_table;
```

▼ WHERE - condition

En SQL, la commande `WHERE` permet d'extraire les lignes d'une table dans une base de données, qui respectent une condition. Cela permet de filtrer les données désirées.

```
SELECT une_colonne FROM une_table WHERE une_condition;
```

Cette requête SQL va donc sélectionner grâce à la commande `SELECT`, la colonne `une_colonne` provenant de la table nommée `une_table` qui respecte la condition `une_condition` de la commande `WHERE`.

Opérateurs de conditions

Opérateur	Description
=	Égal
!= ou <>	Différent
>	Strictement supérieur
>=	Supérieur ou égal
<	Strictement inférieur
<=	Inférieur ou égal
IS NULL	Valeur est égale à NULL
IS NOT NULL	Valeur n'est pas égale à NULL

Exemple concret :

```
SELECT * FROM clients WHERE ville = 'Grenoble'; # Selectionne tous les clients qui
SELECT * FROM clients WHERE ville != 'Grenoble'; # Selectionne tous les clients qui
SELECT * FROM clients WHERE command > 3; # Selectionne tous les clients qui ont plu
SELECT * FROM clients WHERE command >= 3; # Selectionne tous les clients qui ont 3
SELECT * FROM clients WHERE command < 3; # Selectionne tous les clients qui ont moi
SELECT * FROM clients WHERE command <= 3; # Selectionne tous les clients qui ont 3
SELECT * FROM clients WHERE email IS NULL; # Selectionne tous les clients qui ont u
SELECT * FROM clients WHERE email IS NOT NULL; # Selectionne tous les clients qui n
```

▼ BETWEEN - condition

En SQL, l'opérateur `BETWEEN` permet de sélectionner des enregistrements en fonction d'une intervalle. Cet opérateur s'utilise avec la commande `WHERE`. L'intervalle peut être constitué de chaînes de caractères, de nombres ou de dates.

```
SELECT une_colonne FROM une_table WHERE une_colonne BETWEEN "valeur_1" AND "valeur_
```

Exemple concret :

```
SELECT * FROM clients WHERE command BETWEEN 0 AND 3; # Selectionne toutes les infos
SELECT nom, commande FROM clients WHERE nom BETWEEN "Kévin" AND "Arthur"; # Selecti
```

▼ LIKE - condition

En SQL, l'opérateur `LIKE` permet de faire une recherche suivant un modèle sur les valeurs d'une colonne. Il existe plusieurs modèles de recherches (pattern en anglais) que nous détaillerons par la suite. Une nouvelle fois, cet opérateur s'utilise avec la commande `WHERE`. C'est un opérateur extrêmement puissant qui offre de nombreuses possibilités.

```
SELECT une_colonne FROM une_table WHERE une_colonne LIKE un_modele;
```

Caractères joker :

L'opérateur `LIKE` est inséparable des caractères jokers (wildcards) : `%` (pourcentage) et `_` (underscore).

Un caractère joker est utilisé pour se substituer à n'importe quel autre caractère, au sein d'une chaîne de caractères :

Joker `%` : il représente aucun, un seul ou plusieurs caractères.

Joker

`_` : il représente un seul et unique caractère.

Modèle	Commentaire
<code>LIKE "a%"</code>	Recherche toutes les chaînes de caractères qui commencent par le caractère a .
<code>LIKE "%a"</code>	Recherche toutes les chaînes de caractères qui terminent par le caractère a .
<code>LIKE "%a%"</code>	Recherche toutes les chaînes de caractères qui contiennent au moins un caractère a .
<code>LIKE "a%b"</code>	Recherche toutes les chaînes de caractères qui commencent par le caractère a et terminent par le caractère b .
<code>LIKE "a_"</code>	Recherche toutes les chaînes de caractères de trois caractères qui commencent par le caractère a .
<code>LIKE "_a"</code>	Recherche toutes les chaînes de caractères qui possèdent le caractère a en deuxième position .

Exemple concret :

```
SELECT * FROM clients WHERE email LIKE "%@yeah.com"; # Sélectionne tous les clients
SELECT * FROM clients WHERE email LIKE "contact@%.com"; # Sélectionne tous les clie
```

▼ AND et OR - additionner les conditions

Les opérateurs logiques `AND` et `OR` peuvent être utilisés en complément de la commande `WHERE` pour combiner des conditions.

```
SELECT une_colonne FROM une_table WHERE une_condition AND une_autre_condition;
SELECT une_colonne FROM une_table WHERE une_condition OR encore_condition;
```

Exemple concret :

```
SELECT * FROM clients WHERE ville = 'Grenoble' AND command > 3;
SELECT nom, prenom FROM clients WHERE command > 2 OR email IS NOT NULL;
```

▼ ORDER BY - trier

En SQL, la commande `ORDER BY` permet de trier les résultats d'une requête. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant (`ASC`) ou descendant (`DESC`).

```
SELECT une_colonne FROM une_table ORDER BY une_colonne_2;
```

Exemple concret :

```
SELECT * FROM client ORDER BY name ASC; # Selectionne toutes les infos clients, tri
SELECT nom, prenom FROM client ORDER BY id DESC; # Selectionne les noms + prénoms d
```

Pour **trier sur plusieurs colonnes**, il faut donc les séparer avec des `,` (virgules).

```
SELECT * FROM client ORDER BY name ASC, id DESC; # Selectionne toutes les infos cli
```

▼ Jointures - fusionner plusieurs tables pour une requête

Depuis le début nos requêtes portent uniquement sur une table à la fois. **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 en même temps** de manière efficace. Dans ce chapitre, nous aborderons deux types de jointures différents avec les commandes SQL suivantes : `INNER JOIN` et `LEFT JOIN`.

Les jointures SQL consistent à associer les lignes de deux tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table. On nomme ce concept : la condition. L'intérêt des jointures ? Elles sont tout d'abord un moyen d'optimiser les processus, en améliorant leur vitesse d'exécution. De manière générale, il est préférable de faire une seule requête avec plusieurs jointures plutôt que plusieurs requêtes simples. Elles permettent également d'éviter les répétitions d'informations.

INNER JOIN

Dans le langage SQL la commande `INNER JOIN`, est un type de **jointure très commune pour lier plusieurs tables entre-elles** dans une même requête. Cette commande **retourne les enregistrements lorsqu'il y a au moins une ligne dans chaque colonne listé dans la condition.** Autrement dit, lorsque la condition est respectée dans les deux tables.

```
SELECT * FROM table_1 INNER JOIN table_2 ON table_1.une_colonne = table_2.autre_col
```

Cette requête SQL va donc sélectionner grâce à la commande `INNER JOIN` les enregistrements des tables `table_1` et `table_2` lorsque les données de la colonne `une_colonne` de la table `table_1` est égal aux données de la colonne `autre_colonne` de la table `table_2`.

Exemple concret :

```
SELECT * FROM client INNER JOIN commandes ON client.commande_id = commande.id; # Se
SELECT * FROM mere INNER JOIN enfant ON mere.enfant_id = enfant_id;
```

LEFT JOIN

Dans le langage SQL la commande `LEFT JOIN`, est un type de jointure commune pour lier plusieurs tables entre-elles dans une même requête. **Cette commande retourne tous les enregistrements de la table première table, celle de gauche (left), avec la correspondance dans la deuxième table si la condition est respectée.** En d'autres termes, ce type de jointure permet de retourner tous les enregistrements d'une table avec les données liées d'une autre table si elles existent

```
SELECT * FROM table_1 LEFT JOIN table_2 ON table_1.une_colonne = table_2.autre_colo
```


Cette requête SQL va donc sélectionner grâce à la commande `LEFT JOIN` tous les enregistrements de la table `table_1` lorsque les données de la colonne `une_colonne` de la table `table_1` est égal aux données de la colonne `autre_colonne` de la table `table_2`.

Exemple concret :

```
SELECT * FROM client LEFT JOIN commandes ON client.commande_id = commande.id;
SELECT * FROM mere LEFT JOIN enfant ON mere.enfant_id = enfant_id;
```

▼ + Create

```
INSERT INTO `utilisateur` (`nom`, `prenom`, `email`)
VALUES ('Durantay', 'Quentin', 'quentin@gmail.com');
```

 Remarquez que je ne me préoccupe pas de l'id. Je pourrais tout à fait le renseigner. Mais souvenez-vous, nous l'avons configuré de manière à ce que MySQL l'auto-incrémente pour nous.

Exemple générique, vous pouvez avoir X colonne et donc X valeurs à passer.

```
INSERT INTO ma_table (`colonne_1`, `colonne_2`, `...`)
VALUES ('valeur_colonne_1', 'valeur_colonne_2', '...');
```

Il est possible d'ajouter plusieurs objets en une seule commande en séparant leurs valeurs par des virgules

```
INSERT INTO `utilisateur` (`nom`, `prenom`, `email`)
VALUES
('Doe', 'John', 'john@yahoo.fr'),
('Smith', 'Jane', 'jane@hotmail.com'),
('Dupont', 'Sebastien', 'sebastien@orange.fr'),
('Martin', 'Emilie', 'emilie@gmail.com');
```

▼ Update

```
UPDATE `utilisateur` SET `email` = 'quentind@gmail.com' WHERE `id` = '1'; # Change l'email
UPDATE `utilisateur` SET `nom` = 'dumet' WHERE `id` = '3'; # Change le nom de l'utilisateur
UPDATE `utilisateur` SET `prenom` = 'kevin' WHERE `email` = 'sf@gmail.com'; # Change le prenom
```

Mot clé	Description
<code>UPDATE table</code>	Signifie à SQL que vous souhaitez mettre à jour de la donnée dans votre BDD. Vous indiquez aussi la table dans laquelle se trouve(nt) le ou les objets que vous souhaitez modifier.
<code>SET une_colonne = une_valeur</code>	Sert à indiquer à SQL quelles sont la ou les colonnes à modifier , et quelles sont la ou les valeurs qu'elles doivent désormais prendre.
<code>WHERE une_colonne = une_valeur</code>	C'est ce qu'on appelle un filtre . Vous allez les voir plus en détail dans la partie 3, mais sachez qu'ils servent à restreindre la commande en cours à un ou des objets répondant à des conditions précises. Ici, on va mettre à jour uniquement l'objet dont l'id est 1, soit le premier utilisateur !

Exemple générique

```
UPDATE une_table SET une_colonne = une_valeur WHERE une_colonne = une_valeur;
```

 **Commande à utiliser avec pré-caution ! ne pas reproduire ce qui suit**

```
UPDATE `utilisateur` SET `prenom` = "quentin";
```

Vous n'avez pas utilisé `WHERE` pour filtrer les entités à modifier, tous vos utilisateurs s'appellent Quentin ...

▼ Delete

```
DELETE FROM `utilisateur` WHERE `id` = '2'; # Supprime l'utilisateur avec l'id 2  
DELETE FROM `utilisateur` WHERE `prenom` = 'Quentin'; # Supprime le/les utilisateur(s)  
DELETE FROM `utilisateur`; # Supprime tous les enregistrements
```

ZONE DANGER

Réfléchissez à deux fois avant d'utiliser les commandes suivantes !

```
DROP TABLE `utilisateur`; # Supprime une table  
DROP DATABASE `test`; # Supprime une base de données
```