

Table des matières

SQL	2
Qu'est-ce que « SQL » ?	2
Quelques règles de base	2
Les bases SQL	3
La commande 'SELECT'	3
La condition 'WHERE'	6
La commande 'INSERT INTO'	8
La commande 'DELETE'	9
La commande 'UPDATE'	10
SQL Avancé	11
Le tri et la commande 'ORDER BY'	11
Les commandes 'GROUP BY' et 'HAVING'	12
a) <i>GROUP BY</i>	12
b) <i>HAVING</i>	13
Caractères génériques	14
Les fonctions	17
Les fonctions d'agrégations	17
Les fonctions personnalisées	19

SQL

Qu'est-ce que « SQL » ?

SQL, pour « Structured Query Language », est le langage standard pour travailler sur les bases de données relationnelles.

La programmation SQL est utilisée pour insérer, rechercher, mettre à jour et supprimer des enregistrements dans une base de données.

Il existe beaucoup de SGBD qui utilisent SQL : MySQL, Oracle, Ms SQL Server etc...

Quelques règles de base

- Le symbole étoile '*' sélectionne toutes les colonnes de la table spécifié.
- Les string (chaines de caractères) doivent être entourés de simple quote.
- Les nombres ne doivent pas être entourés de simple ou double quote.
- Les données de type 'date' doivent être entouré par des simple quote dans le format 'YYYY-MM-DD'.

Les bases SQL

La commande 'SELECT'

La commande SELECT va permettre de récupérer des données à partir des tables de la base de données. Cela fait partie du langage de manipulation de données chargé d'interroger les données de la base de données.

C'est la commande SQL la plus utilisée, voici sa syntaxe :

```
SELECT [DISTINCT|ALL] {*|[fieldExpression[AS newName]]}  
FROM tableName [alias]  
[WHERE condition]  
[GROUP BY fieldName]  
[HAVING condition]  
[ORDER BY fieldName]
```

- **SELECT** est le mot-clé SQL qui dit qu'on veut récupérer des données. A la suite de SELECT, voici les possibilités :
 - **[DISTINCT | ALL]** sont des mots-clés optionnels. DISTINCT permet d'éviter les redondances dans les résultats (pas de doublons). La valeur par défaut est ALL.
 - **{* | fieldExpression [AS newName]}** : l'étoile '*' sélectionne toutes les colonnes de la table spécifié. 'fieldExpression' effectue certains calculs sur les champs spécifiés, par exemple l'ajout de nombres la somme ou autre.
- **FROM tableName** est le mot-clé qui permet de spécifier la ou les tables sur lesquelles récupérer les données. Il faut au minimum indiqué une table. Les tables doivent être séparés par une virgule. Il est possible de donner un alias à une table.
- **WHERE** est optionnel. Ce mot-clé peut être utilisé pour spécifié des critères lors de la recherche des données.

- **GROUP BY** est utilisé pour assembler des enregistrements ayant les mêmes valeurs de champ.
- **HAVING** est utilisé pour spécifier des conditions lors de l'utilisation de 'GROUP BY'.
- **ORDER BY** permet de trier les résultats.

Exemples d'utilisation de la commande SELECT :

Pour la suite, nous allons partir d'une base de données existante.
Cette base de données contient uniquement deux tables : une table « directors » et une table « movies ».

Voici un aperçu du contenu de ces deux tables.

Table réalisateur :

director_id	name	date_of_birth	gender
1	Steven Spielberg	1946-12-18	Male
2	George Lucas	1944-05-14	Male

Table film :

movie_id	title	year_released	director_id
1	E.T L'extraterrestre	1982-10-02	1
2	Ready Player One	2018-01-01	1
3	Star Wars 7	2015-12-04	2
4	Les infiltrés	2006-01-01	3

- Pour récupérer et afficher uniquement le titre et la date de sortie de tous mes films, j'utilise cette commande :

```
SELECT title, year_released  
FROM movies
```

- Pour récupérer toutes les colonnes de ma table 'movies' :

```
SELECT *  
FROM movies
```

- Imaginons que l'on veut le titre du film et sa date de sortie dans un seul champ. On pourrait faire comme ceci :

```
SELECT Concat(title, ' (' , year_released,  
' )') AS Concat  
FROM movies
```

- Afficher les réalisateurs avec leur année de naissance uniquement :

```
SELECT name, LEFT(`date_of_birth`,4) AS  
`year_of_birth`  
FROM directors
```

La condition 'WHERE'

La commande WHERE va permettre de restreindre certains résultats à une condition bien particulière.

Voici une syntaxe de base utilisant WHERE :

```
SELECT *  
FROM movies  
WHERE director_id = 1
```

Cette commande va permettre de récupérer tous les films qui ont été réalisés par le réalisateur dont l'id est 1.

On peut combiner plusieurs conditions grâce aux opérateurs logiques.

Par exemple, on ne veut que les films dont l'id réalisateur est 1 et l'année de sortie est supérieure à 2000 :

```
SELECT *  
FROM movies  
WHERE director_id = 1 AND year_released >  
'2000'
```

La clause WHERE, lorsqu'elle est utilisée avec le mot-clé IN, affecte uniquement les lignes dont les valeurs correspondent à la liste de valeurs fournie dans le mot clé IN.

Par exemple :

```
SELECT *  
FROM movies  
WHERE director_id IN (1,2,5)
```

On peut également exclure certains résultats directement avec le mot-clé NOT IN

Par exemple :

```
SELECT *  
FROM movies  
WHERE director_id NOT IN (1,2,5)
```

La commande 'INSERT INTO'

SQL utilise la commande INSERT pour stocker des données dans une table. La commande INSERT crée une nouvelle ligne dans la table pour stocker les données.

Voici la syntaxe de la commande INSERT :

```
INSERT INTO tableName(column1, column2,...)
VALUES(value1, value2, ...)
```

- **INSERT INTO tableName** est la commande qui va spécifié dans quelle table ajouter le nouvel enregistrement.
- **(column1, column2,...)** spécifie les colonnes qui vont être mis à jour dans le nouvel enregistrement.
- **(value1, value2,...)** spécifie les valeurs à mettre à jour dans le nouvel enregistrement.

Exemples d'utilisation de la commande 'INSERT' :

```
INSERT INTO directors(name, date_of_birth,
gender)
VALUE ('Martin Scorsese', '1942-11-17', 'Male')
```

L'ordre des colonnes n'a aucun effet sur la requête INSERT tant que les valeurs correctes ont été mappées aux colonnes correctes.

Par défaut, MySQL insère des valeurs NULL dans les colonnes ignorées dans la requête INSERT.

La commande INSERT peut également être utilisée pour insérer des données dans une table à partir d'une autre table. La syntaxe de base est la suivante :

```
INSERT INTO table_1 SELECT * FROM table_2
```


La commande 'DELETE'

La commande SQL 'DELETE' permet de supprimer les lignes qui ne sont plus requises des tables de la base de données. Il supprime toute la ligne de la table. La commande DELETE peut supprimer plusieurs lignes d'une table dans une même requête.

Une fois qu'une ligne a été supprimée, elle ne peut pas être récupérée. Il est donc vivement recommandé d'effectuer des sauvegardes de la base de données avant de supprimer des données de celle-ci. Cela peut vous permettre de restaurer la base de données et d'afficher les données ultérieurement si nécessaires.

Voici la syntaxe de la commande DELETE :

```
DELETE FROM tableName  
[WHERE Condition]
```

Si la clause WHERE n'est pas spécifiée, toutes les lignes de la table seront supprimées.

Je veux supprimer tous les films dont le réalisateur est George Lucas :

```
DELETE FROM movies WHERE director_id = 2
```

La commande 'UPDATE'

La commande SQL 'UPDATE' est utilisée pour mettre à jour des lignes d'une table de la base de données. La commande 'UPDATE' peut être utilisée pour mettre à jour un ou plusieurs champs à la fois. Il peut également être utilisé pour mettre à jour une table avec les valeurs d'une autre table.

Voici la syntaxe de la commande UPDATE :

```
UPDATE tableName  
SET columnName = newValue  
[WHERE condition]
```

- **UPDATE tableName** est la commande qui va spécifier dans quelle table mettre à jour l'enregistrement.
- **SET columnName = newValue** est le nom et la valeur du champ à mettre à jour.

Exemple :

```
UPDATE directors  
SET name = 'Steven Allan Spielberg'  
WHERE name = 'Steven Spielberg'
```

SQL Avancé

Le tri et la commande 'ORDER BY'

Dans cette section, nous verrons comment trier les résultats de notre requête. Le tri consiste simplement à réorganiser nos résultats de requête d'une manière spécifiée. Le tri peut être effectué sur une seule colonne ou sur plusieurs colonnes. Cela peut être fait sur les nombres, les chaînes de caractères ainsi que les types de données date.

La clause 'ORDER BY' permet de trier les ensembles de résultats de la requête par ordre croissant ou décroissant. Il est utilisé conjointement avec la requête SELECT.

Voici sa syntaxe :

```
SELECT *  
FROM tableName  
ORDER BY fieldName(s) [ASC | DESC]
```

ASC permet de trier dans l'ordre croissant.

DESC permet de trier dans l'ordre décroissant.

Par défaut, c'est **ASC** qui est utilisé.

Exemple :

Je veux pouvoir récupérer la liste de mes films triés par date de sortie (du plus récent au plus vieux) :

```
SELECT *  
FROM movies  
ORDER BY year_released DESC
```

Les commandes 'GROUP BY' et 'HAVING'

a) *GROUP BY*

La clause GROUP BY est une commande SQL utilisée pour regrouper des lignes ayant les mêmes valeurs.

Les requêtes contenant la clause GROUP BY sont appelées requêtes groupées et ne renvoient qu'une seule ligne pour chaque élément groupé.

Voici sa syntaxe :

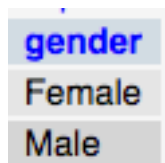
```
SELECT statement...  
GROUP BY columnName(s)
```

Exemple :

- Parmi les réalisateurs, je veux connaître les valeurs uniques possible pour l'attribut 'gender' :

```
SELECT gender  
FROM movies  
GROUP BY gender
```

Résultat :



gender
Female
Male

- On pourrait également choisir de grouper les films par réalisateur ET date de sortie.

```
SELECT director_id, year_released  
FROM movies  
GROUP BY director_id, year_released
```

Si le réalisateur est le même mais que l'année de sortie diffère, une ligne sera rajoutée. En revanche, si le réalisateur ET la date de sortie sont les mêmes, ils seront regroupés sur une seule ligne.

- Il est possible d'utiliser la clause 'GROUP BY' pour utiliser les fonctions d'agrégations. Par exemple, je veux connaître le nombre de réalisateur du genre masculin et féminin.

```
SELECT gender, COUNT(director_id)
FROM directors
GROUP BY gender
```

b) *HAVING*

Dans certains cas, nous voudrions restreindre les résultats renvoyés par la clause GROUP BY.

Dans de tels cas, nous utiliserons la clause HAVING.

Supposons que nous voulions connaître tous les films pour le réalisateur 1 :

```
SELECT *
FROM movies
GROUP BY director_id, year_released
HAVING director_id = 1
```

Caractères génériques

Les caractères génériques sont des caractères qui permettent de rechercher des données correspondant à des critères complexes. Les caractères génériques sont utilisés conjointement avec l'opérateur de comparaison LIKE ou l'opérateur de comparaison NOT LIKE.

a) *LIKE*

L'opérateur LIKE est utilisé dans la clause WHERE des requêtes SQL. Ce mot-clé permet d'effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre.

Voici la syntaxe :

```
SELECT *  
FROM tableName  
WHERE columnName LIKE modele
```

b) NOT LIKE

L'opérateur **NOT LIKE** s'utilise de la même manière que **LIKE** mais retourne le résultat inverse.

c) Le pourcentage %

Le caractère pourcentage est utilisé pour spécifier un motif de zéro (0) caractère ou plus. Il a la syntaxe de base suivante :

```
SELECT statement
FROM table
WHERE fieldName LIKE 'xxx%'
```

Par exemple :

- Chercher tous les réalisateurs dont le nom commence par 'Steven' :

```
SELECT *
FROM directors
WHERE name LIKE 'Steven%'
```

- Chercher tous les réalisateurs dont le nom contient la lettre 's' :

```
SELECT *
FROM directors
WHERE name LIKE '%s%'
```

d) Underscore _

Le caractère underscore est utilisé pour faire correspondre exactement un caractère.

Il a la syntaxe de base suivante :

```
SELECT statement
FROM table
WHERE fieldName LIKE 'xxx_'
```

Supposons que nous voulons rechercher tous les films sortis au cours des années 200x où x est exactement un caractère qui pourrait avoir n'importe quelle valeur :

```
SELECT *  
FROM movies  
WHERE year_released LIKE '200_'
```

e) ESCAPE

Le mot clé ESCAPE est utilisé pour échapper aux caractères spéciaux ou génériques tels que le pourcentage (%) et l'underscore (_) s'ils font partie des données.

Par exemple, nous voulons chercher le film 'Dix%', voici la requête à utiliser :

```
SELECT *  
FROM movies  
WHERE title LIKE 'Dix%' ESCAPE '%'
```


Les fonctions

Les fonctions d'agrégations

Les fonctions d'agrégation permettent de produire facilement des données résumées à partir de notre base de données.

Il y'a plusieurs fonctions d'agrégations dans SQL :

a) COUNT

La fonction COUNT renvoie le nombre total de valeurs dans le champ spécifié. Cela fonctionne à la fois sur les types de données numériques et non numériques. Toutes les fonctions d'agrégation excluent par défaut les valeurs NULL avant de travailler sur les données.

COUNT (*) est une implémentation spéciale de la fonction COUNT qui renvoie le nombre de toutes les lignes d'une table spécifiée. COUNT (*) prend également en compte les valeurs Null et duplicata.

Par exemple, pour compter le nombre total de films dans ma base de données :

```
SELECT COUNT(movie_id)
FROM movies
```

b) MIN

La fonction MIN retourne la plus petite valeur spécifiée dans la table.

Par exemple, on veut connaître le plus vieux film de notre base de données :

```
SELECT MIN(year_released)
FROM movies
```

c) MAX

La fonction MAX est l'inverse de la fonction MIN : elle retourne la plus grande valeur spécifiée dans la table.

Par exemple, on veut connaître le film le plus récent de notre base de données :

```
SELECT MAX(year_released)
FROM movies
```

d) SUM

La fonction SUM renvoie la somme de toutes les valeurs d'une colonne spécifiée. SUM fonctionne uniquement sur les champs numériques.

Par exemple, si on veut connaître la somme de paiements effectuées :

```
SELECT SUM(paiement)
FROM paiements
```

e) AVG

La fonction AVG renvoie la moyenne des valeurs d'une colonne spécifiée.

Par exemple, si on veut connaître la moyenne d'âge des contacts :

```
SELECT AVG(age)
FROM contacts
```

Les fonctions personnalisées

En SQL, il est possible de créer soi-même ses propres fonctions.
Nous ne verrons pas cette partie dans ce cours.