Support SQL

Le **SQL** (Structured Query Language) est un langage permettant de communiquer avec une base de données. Ce langage informatique est notamment très utilisé par les développeurs web pour communiquer avec les données d'un site web. SQL.sh recense des cours de SQL et des explications sur les principales commandes pour lire, insérer, modifier et supprimer des données dans une base.

Se connecter à SQL avec la ligne de commande

Lancer XAMPP puis SHELL

Pour se connecter avec la ligne de commande à SQL il faut lancer XAMPP et lancer la console SQL depuis XAMPP.

Modifier l'apparence de SHELL

Une fois connecté à SHELL de XAMPP on peut en faisant un clic droit sur la fenêtre > propriétés > modifier l'aspect de la fenêtre la taille de la police etc. Ce qui est souhaitable car on lit un peu mieux.

Se connecter à SQL

```

```

Chercher les commandes notez les commandes de base

```
help
```

```
List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear the current input statement.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
```

```
help    (\h) Display this help.
notee  (\t) Don't write into outfile.
print    (\p) Print current command.
prompt          (\R) Change your mysql prompt.
quit    (\q) Quit mysql.
rehash          (\#) Rebuild completion hash.
source          (\.) Execute an SQL script file. Takes a file name as an argument.
status (\s) Get status information from the server.
tee     (\T) Set outfile [to_outfile]. Append everything into given outfile.
use     (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing binlog with
multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.

For server side help, type 'help contents'
```

Les commandes

Commandes de base
SELECT permet d'afficher une commande de base c'est notre première commande de base et elle va chercher les contenus

```
SELECT 'Bonjour Coco !';
```

SQL peut faire des opérations de base cf. le cours open classroom VOIR reporter ici les exemples

Caractères spéciaux

\n	retour à la ligne
\t	tabulation
\	antislash (eh oui, il faut échapper le caractère d'échappement…)
%	pourcent (vous verrez pourquoi plus tard)
_	souligné (vous verrez pourquoi plus tard aussi)

Je vous l'ai dit, il est nécessaire, pour tout fichier/programme/..., de préciser l'encodage utilisé. Je vous propose de choisir l'UTF-8.
A chaque connexion à MySQL, exécutez donc la commande suivante :

```
SET NAMES 'utf8';
```

- MySQL peut s'utiliser en ligne de commande ou avec une interface graphique.
- Pour se connecter à MySQL en ligne de commande, on utilise :mysql -u utilisateur [-h hôte] -p.
- Pour terminer une instruction SQL, on utilise le caractère ;.
- En SQL, les chaînes de caractères doivent être entourées de guillemets simples '.
- Lorsque l'on se connecte à MySQL, il faut définir l'encodage utilisé, soit directement dans la connexion avec l'option--default-character-set, soit avec la commande SET NAMES.

CREATE DATABASE[1]

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.

Syntaxe

Pour créer une base de données qui sera appelé « ma_base » il suffit d'utiliser la requête suivante qui est très simple:

```
CREATE DATABASE ma_base;
```

Base du même nom qui existe déjà

Avec MySQL, si une base de données porte déjà ce nom, la requête retournera une erreur. Pour éviter d'avoir cette erreur, il convient d'utiliser la requête suivante pour MySQL:

```
CREATE DATABASE IF NOT EXISTS ma_base
```

[1] A chaque moment ou l'on insère faire un aparté sur les types de données cf. https://openclassrooms.com/fr/courses/1959476-administrez-vos-bases-de-donnees-avec-mysql/1960456-les-types-de-donnees#/id/r-1977731

L'option IF NOT EXISTS permet juste de ne pas retourner d'erreur si une base du même nom existe déjà. La base de données ne sera pas écrasée.

Options

Dans le standard SQL la commande CREATE DATABASE n'existe normalement pas. En conséquent il revient de vérifier la documentation des différents SGBD pour vérifier les syntaxes possibles pour définir des options. Ces options permettent selon les cas, de définir les jeux de caractères, le propriétaire de la base ou même les limites de connexion.

➔ 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[2] qui seront contenus dans chacun des colonne (entier, chaîne de caractères, date, valeur binaire …).

Conseils

- **Noms de tables et de colonnes**
- **Soyez cohérents**

Syntaxe

La syntaxe générale pour créer une table est la suivante :

```
CREATE TABLE nom_de_la_table(colonne1 type_donnees, colonne2 type_donnees, colonne3 type_donnees, colonne4 type_donnees)
```

Dans cette requête, 4 colonnes ont été définies. Le mot-clé « type_donnees » sera à remplacer par un mot-clé pour définir le type de données (INT, DATE, TEXT …).
Pour chaque colonne, il est également possible de définir des options telles que (liste non-exhaustive):

- **NOT NULL :** empêche d'enregistrer une valeur nulle pour une colonne.
- **DEFAULT :** attribuer une valeur par défaut si aucune données n'est indiquée pour cette colonne lors de l'ajout d'une ligne dans la table.
- **PRIMARY KEY :** indiquer si cette colonne est considérée comme clé primaire pour un index.

[2] cf. note 1

Exemple

Imaginons que l'on souhaite créer une table utilisateur, dans laquelle chaque ligne correspond à un utilisateur inscrit sur un site web. La requête pour créer cette table peut ressembler à ceci:

```
CREATE TABLE utilisateur
(
    id INT PRIMARY KEY NOT NULL,
    nom VARCHAR(100),
    prenom VARCHAR(100),
    email VARCHAR(255),
    date_naissance DATE,
    pays VARCHAR(255),
    ville VARCHAR(255),
    code_postal VARCHAR(5),
    nombre_achat INT
)
```

Voici des explications sur les colonnes créées :

- **id :** identifiant unique qui est utilisé comme clé primaire et qui n'est pas nulle
- **nom :** nom de l'utilisateur dans une colonne de type VARCHAR avec un maximum de 100 caractères au maximum
- **prenom :** idem mais pour le prénom
- **email :** adresse email enregistré sous 255 caractères au maximum
- **date_naissance :** date de naissance enregistré au format AAAA-MM-JJ (exemple : 1973-11-17)
- **pays :** nom du pays de l'utilisateur sous 255 caractères au maximum
- **ville :** idem pour la ville
- **code_postal :** 5 caractères du code postal
- **nombre_achat :** nombre d'achat de cet utilisateur sur le site

VOIR ajouter AUTO_INCREMENT au champs de la clef primaire !!!

➢ SELECT

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

Commande basique

L'utilisation basique de cette commande s'effectue de la manière suivante:

```
SELECT nom_du_champ FROM nom_du_tableau;
```

Cette requête SQL va **sélectionner** (SELECT) le champ « nom_du_champ » **provenant** (FROM) du tableau appelé « nom_du_tableau ».

Exemple

Imaginons une base de données appelée « client » qui contient des informations sur les clients d'une entreprise.
Table « client » :

Id	nom	prenom	ville
1	**Dupont**	**Pierre**	**Paris**
2	**Gauriau**	**Marie-Hélène**	**Paris**
3	**Jaadar**	**Houda**	**Villeneuve**
4	**Diarra**	**Abdoulaye**	**Franconville**
5	**Martin**	**Jean**	**Marseille**

Si l'on veut avoir la liste de toutes les villes des clients, il suffit d'effectuer la requête SQL ci-dessous :
SELECT ville FROM client
De cette manière on obtient le résultat suivant :

ville
Paris
Paris
Villeneuve
Franconville
Marseille

Obtenir plusieurs colonnes

Avec la même table client il est possible de lire plusieurs colonnes à la fois. Il suffit tout simplement de séparer les noms des champs souhaités par une virgule. Pour obtenir les **prénoms** et les **noms** des **clients** il faut alors faire la requête suivante:

```
SELECT prenom, nom FROM client;
```

Ce qui retourne ce résultat:

nom	prenom
Dupont	Pierre
Gauriau	Marie-Hélène
Jaadar	Houda
Diarra	Abdoulaye
Martin	Jean

Obtenir toutes les colonnes d'un tableau

Il est possible de retourner automatiquement toutes les colonnes d'un tableau sans avoir à connaître le nom de toutes les colonnes. Au lieu de lister toutes les colonnes, il faut simplement utiliser le caractère « * » (étoile). C'est un joker qui permet de sélectionner toutes les colonnes. Il s'utilise de la manière suivante:

```
SELECT * FROM client;
```

Cette requête SQL retourne exactement les mêmes colonnes qu'il y a dans la base de données. Dans notre cas, le résultat sera donc:

Id	nom	prenom	ville
1	Dupont	Pierre	Paris
2	Gauriau	Marie-Hélène	Paris
3	Jaadar	Houda	Villeneuve
4	Diarra	Abdoulaye	Franconville

5	Martin	Jean	Marseille

Il y a des avantages et des inconvénient à l'utiliser.

Avantages du caractère étoile

Ce caractère peut lister toutes les colonnes sans avoir besoin de connaître leurs noms. C'est très pratique pendant une phase de tests ou de debug pour voir ce que contient un tableau rapidement et facilement.
Une requête utilisant ce caractère à moins de chance d'échouer si les tables évoluent. Puisque les noms des champs ne sont pas spécifiés dans la sélection, la requête continuera de fonctionner si un nom de champ est modifié ou si une colonne est supprimée. A l'inverse, une requête qui spécifie la lecture des champs « nom » et « ville » échouera si le champ « ville » est remplacé par le champ « code_postal ».

Inconvénients du caractère étoile

Bien que ce caractère semble magique pour les débutants, il y a tout de même des inconvénients pour l'utiliser dans une application. Son utilisation implique que le système de gestion de base de données retourne toutes les colonnes. Or, si une application désire seulement le prénom et le nom du client, il est inutile de lire d'autres données telles que la ville.
Utilisé à mauvais escient, le caractère étoile peut avoir de mauvaises conséquences sur les performances d'une application. Pour optimiser les performances, il convient de lire seulement les données qui seront utilisées par l'application.
Par ailleurs, lorsque le sélecteur étoile est utilisé, les colonnes sont retournées dans le même ordre qu'elles sont présentes en base. Si une application utilise le champ étoile et qu'elle lit les informations en fonction de l'ordre dans lequel les colonnes sont retournées, il y aura une grosse erreur si la base de données est modifiée (modification d'un champ, modification de l'ordre des champs …).

Ne pas utiliser le sélecteur étoile dans une application

Malgré les avantages présentés, la bonne pratique consiste à ne pas utiliser ce caractère dans une application. Il a été précisé qu'une requête est plus **évolutive** en utilisant ce caractère dans le cas où des champs sont modifiés.

Mais en réalité, une application doit être adaptée s'il y a un gros changement dans la base de données.
Ce caractère doit par conséquent être réservé à un usage ponctuel lorsqu'un développeur souhaite lire rapidement le contenu d'une base.

Requête avancé : ordre des commandes du SELECT

Cette commande SQL est relativement commune car il est très fréquent de devoir lire les données issues d'une base de données. Il existe plusieurs commandes qui permettent de mieux gérer les données que l'on souhaite lire. Voici un petit aperçu des fonctionnalités possibles qui sont abordées sur le reste du site:

- Joindre un autre tableau aux résultats
- Filtrer pour ne sélectionner que certains enregistrements
- Classer les résultats
- Grouper les résultats pour faire uniquement des statistiques (note moyenne, prix le plus élevé …)

Un requête SELECT peut devenir assez longue. Juste à titre informatif, voici une requête SELECT qui possède presque toutes les commandes possibles:

```
SELECT * FROM table WHERE condition GROUP BY expression HAVING condition { UNION | INTERSECT | EXCEPT } ORDER BY expression LIMIT count OFFSET start;
```

A noter : cette requête imaginaire sert principale d'aide-mémoire pour savoir dans quel ordre sont utilisé chacunes des commandes au sein d'une requête SELECT.

➔ INSERT INTO

L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup.

Insertion d'une ligne à la fois

Pour insérer des données dans une base, il y a 2 syntaxes principales :

- Insérer une ligne en indiquant les informations pour chaque colonne existante (en respectant l'ordre)
- Insérer une ligne en spécifiant les colonnes que vous souhaitez compléter. Il est possible d'insérer une ligne renseignant seulement une partie des colonnes

Insérer une ligne en spécifiant toutes les colonnes

La syntaxe pour remplir une ligne avec cette méthode est la suivante :

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

Cette syntaxe possède les avantages et inconvénients suivants :

- Obligé de remplir toutes les données, tout en respectant l'ordre des colonnes
- Il n'y a pas le nom de colonne, donc les fautes de frappe sont limitées. Par ailleurs, les colonnes peuvent être renommées sans avoir à changer la requête
- L'ordre des colonnes doit rester identique sinon certaines valeurs prennent le risque d'être complétée dans la mauvaise colonne

Insérer une ligne en spécifiant seulement les colonnes souhaitées

Cette deuxième solution est très similaire, excepté qu'il faut indiquer le nom des colonnes avant « VALUES ». La syntaxe est la suivante :

```
INSERT INTO table (nom_colonne_1, nom_colonne_2, ...
 VALUES ('valeur 1', 'valeur 2', ...);
```

A noter : il est possible de ne pas renseigner toutes les colonnes. De plus, l'ordre des colonnes n'est pas important.

Insertion de plusieurs lignes à la fois

Il est possible d'ajouter plusieurs lignes à un tableau avec une seule requête. Pour ce faire, il convient d'utiliser la syntaxe suivante :

```
INSERT INTO client (nom, prenom, ville) VALUES
 (Pisola', 'Patrick', 'Paris'),
 (Diallo', 'Alpha', 'Dunkerque'),
 ('Don', 'Juan', 'Log Angeles'),
 ('Igoudjil', 'Idir', 'Marseille');
```

A noter: lorsque le champ à remplir est de type VARCHAR ou TEXT il faut indiquer le texte entre guillemet simple. En revanche, lorsque la colonne est un numérique tel que INT ou BIGINT il n'y a pas besoin d'utiliser de guillemet, il suffit juste d'indiquer le nombre.

Un tel exemple sur une table vide va créer le tableau suivant :

Id	nom	prenom	ville
1	Dupont	Pierre	Paris
2	Gauriau	Marie-Hélène	Montpellier
3	Jaadar	Houda	Tinghir
4	Diarra	Abdoulaye	St-Louis
5	Martin	Jean	Marseille
6	Pisola	Patrick	Paris
7	Diallo	Alpha	Dunkerque
8	Don	Juan	Los Angeles
9	Igoudjil	Idir	Marseille

➔ **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.

Syntaxe

La syntaxe basique d'une requête utilisant UPDATE est la suivante:

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

Cette syntaxe permet d'attribuer une nouvelle valeur à la colonne nom_colonne_1 pour les lignes qui respectent la condition stipulé avec WHERE. Il est aussi possible d'attribuer la même valeur à la colonne nom_colonne_1 pour toutes les lignes d'une table si la condition WHERE n'était pas utilisée.

A noter, pour spécifier en une seule fois plusieurs modification, il faut séparer les attributions de valeur par des virgules. Ainsi la syntaxe deviendrait la suivante :

```
UPDATE table SET colonne_1 = 'valeur 1', colonne_2 = 'valeur 2', colonne_3 = 'valeur 3' WHERE condition;
```

Exemple

Imaginons une table « client » qui présente les coordonnées de clients.

Table « client » :

Id	nom	prenom	ville
1	Dupont	Pierre	Paris
2	Gauriau	Marie-Hélène	Montpellier
3	Jaadar	Houda	Tinghir
4	Diarra	Abdoulaye	St-Louis
5	Martin	Jean	Marseille
6	Pisola	Patrick	Paris
7	Diallo	Alpha	Dunkerque
8	Don	Juan	Los Angeles
9	Igoudjil	Idir	Marseille

Modifier une ligne

Pour modifier le client Abdoulaye, il est possible d'utiliser la requête SQL suivante :

```
UPDATE client SET prenom = 'Samba',  ville = 'Nouakchott',
WHERE id = 4;
```

Cette requête sert à définir la colonne prénom à «Samba», la ville à «Nouakchott» uniquement pour ligne où l'identifiant est égal à 4.
Résultats :

Id	nom	prenom	ville
1	Dupont	Pierre	Paris
2	Gauriau	Marie-Hélène	Montpellier
3	Jaadar	Houda	Tinghir

4	Diarra	Samba	Nouakchott
5	Martin	Jean	Marseille
6	Pisola	Patrick	Paris
7	Diallo	Alpha	Dunkerque
8	Don	Juan	Los Angeles
9	Igoudjil	Idir	Marseille

Modifier toutes les lignes

Il est possible d'effectuer une modification sur toutes les lignes en omettant d'utiliser une clause conditionnelle. Il est par exemple possible de mettre la valeur « Paris » dans la colonne « ville» pour toutes les lignes de la table, grâce à la requête SQL ci-dessous.

```
UPDATE client SET ville = 'Paris';
```

Résultats :

Id	nom	prenom	ville
1	Dupont	Pierre	Paris
2	Gauriau	Marie-Hélène	Paris
3	Jaadar	Houda	Paris
4	Diarra	Samba	Paris
5	Martin	Jean	Paris
6	Pisola	Patrick	Paris
7	Diallo	Alpha	Paris
8	Don	Juan	Paris

9	Igoudjil	Idir	Paris

➔ **WHERE**

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées.

Syntaxe

La commande WHERE s'utilise en complément à une requête utilisant SELECT. La façon la plus simple de l'utiliser est la suivante:

```
SELECT nom_colonnes FROM nom_table WHERE condition;
```

Exemple

Imaginons une base de données appelée « client » qui contient le nom des clients, le nombre de commandes qu'ils ont effectués et leur ville:

Id	nom	prenom	ville
1	Dupont	Pierre	Paris
2	Gauriau	Marie-Hélène	Montpellier
3	Jaadar	Houda	Tinghir
4	Diarra	Samba	Nouakchott
5	Martin	Jean	Marseille
6	Pisola	Patrick	Paris
7	Diallo	Alpha	Dunkerque
8	Don	Juan	Los Angeles

9	Igoudjil	Idir	Marseille

Pour obtenir seulement la liste des clients qui habitent à Paris, il faut effectuer la requête suivante:

```
SELECT * FROM client WHERE ville = 'paris';
```

Cette requête retourne le résultat suivant:

Id	nom	prenom	ville
1	Dupont	Pierre	Paris
6	Pisola	Patrick	Paris

Attention: dans notre cas tout est en minuscule donc il n'y a pas eu de problème. Cependant, si un table est sensible à la casse, il faut faire attention aux majuscules et minuscules.

Opérateurs de comparaisons

Il existe plusieurs opérateurs de comparaisons. La liste ci-jointe présente quelques uns des opérateurs les plus couramment utilisés.

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à

>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
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

Attention: il y a quelques opérateurs qui n'existe pas dans des vieilles versions de système de gestion de bases de données (SGBD(Système de Gestion de Base de Données)). De plus, il y a de nouveaux opérateurs non indiqués ici qui sont disponibles avec certains SGBD. N'hésitez pas à consulter la documentation de MySQL, PostgreSQL ou autre pour voir ce qu'il vous est possible de faire.

➔ **DELETE**

La commande DELETE en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associé à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées.

Attention

Avant d'essayer de supprimer des lignes, il est recommandé d'effectuer une **sauvegarde de la base de données**, ou tout du moins de la table concernée par la suppression. Ainsi, s'il y a une mauvaise manipulation il est toujours possible de restaurer les données.

Syntaxe

La syntaxe pour supprimer des lignes est la suivante :

```
DELETE FROM `table` WHERE condition;
```

Attention : s'il n'y a pas de condition WHERE alors **toutes** les lignes seront supprimées et la table sera alors vide.

Exemple

Imaginons une table « utilisateur » qui contient des informations sur les utilisateurs d'une application.
Table « utilisateur » :

Id	nom	prenom	ville	date_inscription
1	Dupont	Pierre	Paris	2018-07-12
2	Gauriau	Marie-Hélène	Montpellier	2018-07-12
3	Jaadar	Houda	Tinghir	2013-07-02
4	Diarra	Samba	Nouakchott	2018-07-12
5	Martin	Jean	Marseille	2018-11-12
6	Pisola	Patrick	Paris	2018-07-12
7	Diallo	Alpha	Dunkerque	2015-07-19
8	Don	Juan	Los Angeles	2018-07-22
9	Igoudjil	Idir	Marseille	2000-07-07

Supprimer une ligne

Il est possible de supprimer une ligne en effectuant la requête SQL suivante:

```
DELETE FROM `utilisateur` WHERE `id` = 1;
```

Une fois cette requête effectuée, la table contiendra les données suivantes :

Id	nom	prenom	ville	date_inscription
2	Gauriau	Marie-Hélène	Montpellier	2018-07-12
3	Jaadar	Houda	Tinghir	2013-07-02
4	Diarra	Samba	Nouakchott	2018-07-12
5	Martin	Jean	Marseille	2018-11-12
6	Pisola	Patrick	Paris	2018-07-12
7	Diallo	Alpha	Dunkerque	2015-07-19
8	Don	Juan	Los Angeles	2018-07-22
9	Igoudjil	Idir	Marseille	2000-07-07

Supprimer plusieurs lignes

Si l'on souhaite supprimer les utilisateurs inscrit avant le 01/07/2018, il va falloir effectuer la requête suivante :

```
DELETE FROM `utilisateur` WHERE `date_inscription` < '2018-07-01'
```

La requête permettra alors de supprimer les utilisateurs «Houda» et «Alpha» et «Idir». La table contiendra alors les données suivantes:

Id	nom	prenom	ville	date_inscription
2	Gauriau	Marie-Hélène	Montpellier	2018-07-12
4	Diarra	Samba	Nouakchott	2018-07-12
5	Martin	Jean	Marseille	2018-11-12
6	Pisola	Patrick	Paris	2018-07-12
8	Don	Juan	Los Angeles	2018-07-22

Il ne faut pas oublier qu'il est possible d'utiliser d'autres conditions pour sélectionner les lignes à supprimer.

Supprimer toutes les données

Pour supprimer toutes les lignes d'une table il convient d'utiliser la commande DELETE sans utiliser de clause conditionnelle.

```
DELETE FROM `utilisateur`;
```

Supprimer toutes les données : DELETE ou TRUNCATE

Pour supprimer toutes les lignes d'une table, il est aussi possible d'utiliser la commande TRUNCATE, de la façon suivante :

```
TRUNCATE TABLE `utilisateur`;
```

Cette requête est similaire. La différence majeure étant que la commande TRUNCATE va réinitialiser l'auto-increment s'il y en a

un. Tandis que la commande DELETE ne réinitialise pas l'auto-incrément.

➔ **ALTER TABLE**

La commande ALTER TABLE en SQL permet de modifier une table existante. Il est ainsi possible d'ajouter une colonne, d'en supprimer une ou de modifier une colonne existante, par exemple pour changer le type.

Syntaxe de base

D'une manière générale, la commande s'utilise de la manière suivante:

```
ALTER TABLE nom_table instruction;
```

Le mot-clé « instruction » ici sert à désigner une commande supplémentaire, qui sera détaillée ci-dessous selon l'action que l'on souhaite effectuer : ajouter, supprimer ou modifier une colonne.

Ajouter une colonne

Syntaxe

L'ajout d'une colonne dans une table est relativement simple et peut s'effectuer à l'aide d'une requête ressemblant à ceci:

```
ALTER TABLE nom_table ADD nom_colonne type_donnees;
```

Exemple

Pour ajouter une colonne qui correspond à une rue sur une table utilisateur, il est possible d'utiliser la requête suivante:

```
ALTER TABLE utilisateur ADD adresse_rue VARCHAR(255);
```

Résultats:

Id	nom	prenom	ville	date_inscription	adresse_rue
1	Dupont	Pierre	Paris	2018-07-12	
2	Gauriau	Marie-Hélène	Montpellier	2018-07-12	
3	Jaadar	Houda	Tinghir	2013-07-02	
4	Diarra	Samba	Nouakchott	2018-07-12	
5	Martin	Jean	Marseille	2018-11-12	
6	Pisola	Patrick	Paris	2018-07-12	
7	Diallo	Alpha	Dunkerque	2015-07-19	
8	Don	Juan	Los Angeles	2018-07-22	
9	Igoudjil	Idir	Marseille	2000-07-07	

Supprimer une colonne

Une syntaxe permet également de supprimer une colonne pour une table. Il y a 2 manières totalement équivalente pour supprimer une colonne:

```
ALTER TABLE nom_table DROP nom_colonne;
```

Ou (le résultat sera le même)

```
ALTER TABLE nom_table DROP COLUMN nom_colonne;
```

Modifier une colonne

Pour modifier une colonne, comme par exemple changer le type d'une colonne, il y a différentes syntaxes selon le SGBD(Système de Gestion de Base de Données).

MySQL

```
ALTER TABLE nom_table MODIFY nom_colonne
type_donnees;
```

PostgreSQL

```
ALTER TABLE nom_table ALTER COLUMN nom_colonne TYPE
type_donnees;
```

Ici, le mot-clé « type_donnees » est à remplacer par un type de données tel que INT, VARCHAR, TEXT, DATE …

Renommer une colonne

Pour renommer une colonne, il convient d'indiquer l'ancien nom de la colonne et le nouveau nom de celle-ci.

MySQL

Pour MySQL, il faut également indiquer le type de la colonne.

```
ALTER TABLE nom_table CHANGE colonne_ancien_nom
colonne_nouveau_nom type_donnees;
```

Ici « type_donnees » peut correspondre par exemple à INT, VARCHAR, TEXT, DATE …

PostgreSQL

Pour PostgreSQL la syntaxe est plus simple et ressemble à ceci (le type n'est pas demandé):

```
ALTER TABLE nom_table RENAME COLUMN
colonne_ancien_nom TO colonne_nouveau_nom;
```

➔ **DROP TABLE**

La commande DROP TABLE en SQL permet de supprimer définitivement une table d'une base de données. Cela supprime en même temps les éventuels index, trigger, contraintes et permissions associées à cette table.

Attention : il faut utiliser cette commande avec attention car une fois supprimée, les données sont perdues. Avant de l'utiliser sur une base importante il peut être judicieux d'effectuer un backup (une sauvegarde) pour éviter les mauvaises surprises.

Syntaxe

Pour supprimer une table « nom_table » il suffit simplement d'utiliser la syntaxe suivante :

```
DROP TABLE nom_table;
```

A savoir : s'il y a une dépendence avec une autre table, il est recommandé de les supprimer avant de supprimer la table. C'est le cas par exemple s'il y a des clés étrangères.

Intérêts

Il arrive qu'une table soit créé temporairement pour stocker des données qui n'ont pas vocation à être ré-utiliser. La suppression d'une table non utilisée est avantageux sur plusieurs aspects :

- **Libérer de la mémoire** et alléger le poids des backups
- **Éviter des erreurs** dans le futur si une table porte un nom similaire ou qui porte à confusion
- Lorsqu'un développeur ou administrateur de base de données découvre une application, il est **plus rapide de comprendre le système** s'il n'y a que les tables utilisées qui sont présente

Exemple de requête

Imaginons qu'une base de données possède une table « client_2009 » qui ne sera plus jamais utilisé et qui existe déjà dans un ancien backup. Pour supprimer cette table, il suffit d'effectuer la requête suivante :

```
DROP TABLE client_2009;
```

L'exécution de cette requête va permettre de supprimer la table.

- **ALLER PLUS LOIN**
  - **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 associé à un OFFSET, c'est-à-dire effectuer un décalage sur le jeu de résultat. Ces 2 clauses permettent par exemple d'effectuer des système de pagination (exemple : récupérer les 10 articles de la page 4).

ATTENTION : selon le système de gestion de base de données, la syntaxe ne sera pas pareil. Ce tutoriel va donc présenter la syntaxe pour **MySQL** et pour **PostgreSQL**.

Syntaxe simple

La syntaxe commune aux principales système de gestion de bases de données est la suivante :

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

Cette requête permet de récupérer seulement les 10 premiers résultats d'une table. Bien entendu, si la table contient moins de 10 résultats, alors la requête retournera toutes les lignes.
Bon à savoir : la bonne pratique lorsque l'on utilise LIMIT consiste à utiliser également la clause ORDER BY pour s'assurer que quoi qu'il en soit ce sont toujours les bonnes données qui sont présentées. En effet, si le système de tri est non spécifié, alors il est en principe inconnu et les résultats peuvent être imprévisible.

Limit et Offset avec PostgreSQL

L'offset est une méthode simple de décaler les lignes à obtenir. La syntaxe pour utiliser une limite et un offset est la suivante :

```
SELECT * FROM table LIMIT 10 OFFSET 5;
```

Cette requête permet de récupérer les résultats 6 à 15 (car l'OFFSET commence toujours à 0). A titre d'exemple, pour récupérer les résultats 16 à 25 il faudrait donc utiliser: **LIMIT 10 OFFSET 15**

A noter : Utiliser **OFFSET 0** reviens au même que d'ommettre l'OFFSET.

Limit et Offset avec MySQL

La syntaxe avec MySQL est légèrement différente :

```
SELECT * FROM table LIMIT 5, 10;
```

Cette requête retourne les enregistrements 6 à 15 d'une table. Le premier nombre est l'OFFSET tandis que le suivant est la limite.
Bon à savoir : pour une bonne compatibilité, MySQL accepte également la syntaxe **LIMIT nombre OFFSET nombre**. En conséquent, dans la conception d'une application utilisant MySQL il est préférable d'utiliser cette syntaxe car c'est potentiellement plus facile de migrer vers un autre système de gestion de base de données sans avoir à réécrire toutes les requêtes.

Performance

Ce dernier chapitre est destiné à un public averti. Il n'est pas nécessaire de le comprendre entièrement, mais simplement d'avoir compris les grandes lignes.
Certains développeur pensent à tort que l'utilisation de LIMIT permet de réduire le **temps d'exécution** d'une requête. Or, le temps d'exécution est sensiblement le même car la requête va permettre de récupérer toutes les lignes (donc temps d'exécution identique) PUIS seulement les résultats définit par LIMIT et OFFSET seront retournés. Au mieux, utiliser LIMIT permet de réduire le **temps d'affichage** car il y a moins de lignes à afficher.

- **GROUP BY**

La commande 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 ventes d'un magasin, il est par exemple possible de liste regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

Syntaxe d'utilisation de GROUP BY

De façon générale, la commande GROUP BY s'utilise de la façon suivante

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

A noter : cette commande doit toujours s'utiliser après la commande WHERE et avant la commande HAVING.

Exemple d'utilisation

Prenons en considération une table « achat » qui résume les ventes d'une boutique :

Id	client	tarif	date
1	Pierre	102	2018-06-23
2	Simon	32	2017-12-25
3	Marie	180	2017-01-29
4	Marie	20	2018-02-28
5	Pierre	1250	2017-11-11

Ce tableau contient une colonne qui sert d'identifiant pour chaque ligne, une autre qui contient le nom du client, le coût de la vente et la date d'achat. Pour obtenir le coût total de chaque client en regroupant les commandes des mêmes clients, il faut utiliser la requête suivante :

```
SELECT client, SUM(tarif) FROM achat GROUP BY client
```

La fonction SUM() permet d'additionner la valeur de chaque tarif pour un même client. Le résultat sera donc le suivant :

client	SUM(tarif)
Pierre	1352

Simon	32
Marie	200

La manière simple de comprendre le GROUP BY c'est tout simplement d'assimiler qu'il va éviter de présenter plusieurs fois les mêmes lignes. C'est une méthode pour éviter les doublons.
Juste à titre informatif, voici ce qu'on obtient de la requête sans utiliser GROUP BY.
Requête :

```
SELECT client, SUM(tarif) FROM achat
```

Résultat :

client	SUM(tarif)
Pierre	102
Simon	32
Marie	180
Marie	20
Pierre	1250

Utilisation d'autres fonctions de statistiques

Il existe plusieurs fonctions qui peuvent être utilisées pour manipuler plusieurs enregistrements, il s'agit des fonctions d'agrégations statistiques, les principales sont les suivantes :

- AVG() pour calculer la moyenne d'un set de valeur. Permet de connaître le prix du panier moyen pour de chaque client
- COUNT() pour compter le nombre de lignes concernées. Permet de savoir combien d'achats a été effectué par chaque client
- **MAX()** pour récupérer la plus haute valeur. Pratique pour savoir l'achat le plus cher
- **MIN()** pour récupérer la plus petite valeur. Utile par exemple pour connaître la date du premier achat d'un client
- **SUM()** pour calculer la somme de plusieurs lignes. Permet par exemple de connaître le total de tous les achats d'un client

Ces petites fonctions se révèlent rapidement indispensable pour travailler sur des données.

- **AVG()**

La fonction d'agrégation AVG() dans le langage SQL permet de calculer une valeur moyenne sur un ensemble d'enregistrement de type numérique et non nul.

Syntaxe

La syntaxe pour utiliser cette fonction de statistique est simple :

```
SELECT AVG(nom_colonne) FROM nom_table
```

Cette requête permet de calculer la note moyenne de la colonne « nom_colonne » sur tous les enregistrements de la table « nom_table ». Il est possible de filtrer les enregistrements concernés à l'aide de la commande WHERE. Il est aussi possible d'utiliser la commande GROUP BY pour regrouper les données appartenant à la même entité.
A savoir : la syntaxe est conforme avec la norme SQL et fonctionne correctement avec tous les Systèmes de Gestion de Base de Données (SGBD), incluant : MySQL, PostgreSQL, Oracle et SQL Server.

Exemple

Imaginons une table « achat » qui représente toutes les ventes sur un site d'e-commerce, dans laquelle on enregistre le montant de l'achat, la date et le nom du client.

Id	client	tarif	date
1	Pierre	102	2018-06-23
2	Simon	47	2017-12-25
3	Marie	18	2017-01-29
4	Marie	20	2018-02-28
5	Pierre	160	2017-11-11

Pour connaitre le montant moyen effectué par chaque client, il est possible d'utiliser une requête qui va utiliser :

- GROUP BY pour regrouper les ventes des mêmes clients
- La fonction AVG() pour calculer la moyenne des enregistrements

La requête sera donc la suivante :

```
SELECT client, AVG(tarif) FROM achat GROUP BY client
```

Le résultat sera le suivant :

client	AVG(tarif)
Pierre	131
Simon	47
Marie	19

- **COUNT()**

  En SQL, la fonction d'agrégation COUNT() permet de compter le nombre d'enregistrement dans une table. Connaître le nombre de lignes dans une table est très pratique dans de nombreux cas, par exemple pour savoir combien d'utilisateurs sont présents dans une table ou pour connaître le nombre de commentaires sur un article.

Syntaxe

Pour connaître le nombre de lignes totales dans une table, il suffit d'effectuer la requête SQL suivante :

```
SELECT COUNT(*) FROM table
```

Il est aussi possible de connaître le nombre d'enregistrement sur une colonne en particulier. Les enregistrements qui possèdent la valeur nul ne seront pas comptabilisé. La syntaxe pour compter les enregistrement sur la colonne « nom_colonne » est la suivante :

```
SELECT COUNT(nom_colonne) FROM table
```

Enfin, il est également possible de compter le nombre d'enregistrement distinct pour une colonne. La fonction ne comptabilise pas les doublons pour une colonne choisie. La syntaxe pour compter le nombre de valeur distincte pour la colonne « nom_colonne » est la suivante :

```
SELECT COUNT(DISTINCT nom_colonne) FROM table
```

A savoir : en général, en terme de performance il est plutôt conseillé de filtrer les lignes avec GROUP BY si c'est possible, puis d'effectuer un COUNT(*).

Exemple

Imaginons une table qui liste les utilisateurs d'un site web d'e-commerce :

id	nom	ville	date_inscription	nb_achat	id_dernier_achat
1	Marie	Paris	2017-06-23	5	24
2	Samba	Marseille	2016-11-29	3	36
3	Alpha	Lyon	2017-12-25	0	NULL
4	Houda	Bordeaux	2017-07-07	1	7
5	Idir	Nantes	2017-10-02	0	NULL

Pour compter le nombre d'utilisateurs total depuis que le site existe, il suffit d'utiliser COUNT(*) sur toute la table :

```
SELECT COUNT(*) FROM utilisateur
```

Résultat :

COUNT(*)
5

Utilisation de COUNT(*) avec WHERE

Pour compter le nombre d'utilisateur qui ont effectué au moins un achat, il suffit de faire la même chose mais en filtrant les enregistrements avec WHERE :

```
SELECT COUNT(*) FROM utilisateur WHERE nombre_achat > 0
```

Résultat :

COUNT(*)
3

Utilisation de COUNT(colonne)

Une autre méthode permet de compter le nombre d'utilisateurs ayant effectué au moins un achat. Il est possible de compter le nombre d'enregistrement sur la colonne « id_dernier_achat ». Sachant que la valeur est nulle s'il n'y a pas d'achat, les lignes ne seront pas comptées s'il n'y a pas eu d'achat. La requête est donc la suivante :

```
SELECT COUNT(id_dernier_achat) FROM utilisateur
```

Résultat :

COUNT(id_dernier_achat)
3

Utilisation de COUNT(DISTINCT colonne)

L'utilisation de la clause DISTINCT peut permettre de connaître le nombre de villes distinctes sur lesquels les visiteurs sont répartis. La requête serait la suivante :

```
SELECT COUNT(DISTINCT ville) FROM utilisateur
```

Sachant qu'il y a 4 villes distinctes (cf. Paris, Marseille, Lyon et Nantes), le résultat se présente comme ceci:

COUNT(DISTINCT ville)
4

- **MAX()**

Dans le langage SQL, la fonction d'agrégation MAX() permet de retourner la valeur maximale d'une colonne dans un set d'enregistrement. La fonction peut s'appliquer à des données numériques ou alphanumériques. Il est par exemple possible de rechercher le produit le plus cher dans une table d'une boutique en ligne.

Syntaxe

La syntaxe de la requête SQL pour retourner la valeur maximum de la colonne « nom_colonne » est la suivante:

```
SELECT MAX(nom_colonne) FROM table
```

Lorsque cette fonctionnalité est utilisée en association avec la commande GROUP BY, la requête peut ressembler à l'exemple ci-dessous:

```
SELECT colonne1, MAX(colonne2) FROM table GROUP BY colonne1
```

Exemple

Imaginons un site d'e-commerce qui possède une table « produit » sur lequel on peut retrouver des produits informatiques.

id	nom	prix
1	clavier	50
2	souris	21
3	écran	120
4	disque dur	150

Connaître la valeur la plus élevée

Pour extraire uniquement le tarif le plus élevé dans la table, il est possible d'utiliser la requête suivante:

```
SELECT MAX(prix) FROM produit
```

Le résultat sera le suivant :

max(prix)
150

Connaître l'enregistrement ayant la valeur maximale

Il est aussi possible d'afficher le nom du produit le plus cher en même temps que d'afficher son prix. Pour cela, il suffit simplement d'utiliser la fonction d'agrégation MAX() tout en sélectionnant les autres enregistrements. La requête SQL est alors la suivante :

```
SELECT id, nom, MAX(prix) FROM produit
```

Le résultat de cette requête sera le suivant :

id	nom	max(prix)
4	disque dur	150

- **MIN()**

La fonction d'agrégation MIN() de SQL permet de retourner la plus petite valeur d'une colonne sélectionnée. Cette fonction s'applique aussi bien à des données numériques qu'à des données alphanumériques.

Syntaxe

Pour obtenir la plus petite valeur de la colonne « nom_colonne » il est possible d'utiliser la requête SQL suivante:

```
SELECT MIN(nom_colonne) FROM table
```

Étant données qu'il s'agit d'une fonction d'agrégation, il est possible de l'utiliser en complément de la commande GROUP BY. Cela permet de grouper des colonnes et de connaître la plus petite valeur pour chaque groupe. La syntaxe est alors la suivante:

```
SELECT colonne1, MIN(colonne2) FROM table GROUP BY colonne1
```

Cette exemple permet de regrouper tous les enregistrements de « colonne1 » de la table et de connaître la plus petite valeur de « colonne2 » pour chacun de ces regroupement.

Exemple

Imaginons la base de données d'une boutique en ligne qui contient des produits divers. Ces produits possède une catégorie, un nom, un prix et la date à laquelle ils ont été ajouté dans le catalogue.
Table produits :

id	categorie	nom	prix	date_ajout
1	informatique	Ordinateur	980	2018-07-27
2	informatique	Imprimante	70	2018-07-27

3	maison	Canapé	450	2018-07-27
4	maison	Aspirateur	200	2018-07-27

Utilisation simple

Pour extraire le prix du produit le moins cher de la catégorie « maison », il est possible d'effectuer la requête SQL ci-dessous:

```
SELECT MIN(prix) FROM `produits` WHERE `categorie` = 'maison'
```

Résultat :

prix
200

Le résultat montre bien que le prix le moins cher est celui de l'aspirateur qui coûte 200€.

Utilisation dans un GROUP BY

Il est possible de connaître la date du premier ajout dans chaque catégorie. Cela permet de savoir l'article le plus vieux dans le catalogue pour chaque thématiques. Cela s'effectue à l'aide de la requête suivante:

```
SELECT `categorie`, MIN(date_ajout) FROM `produits` GROUP BY `categorie`
```

Résultats :

categorie	**MIN(date_ajout)**
informatique	2018-07-27
maison	2018-07-27

Le résultat montre bien que les enregistrements de la table sont regroupé par catégorie et que seul la valeur min de « date_ajout » est extrait. Il est aussi possible de connaître la date du dernier ajout de chaque catégorie en utilisant la fonction MAX().

- **SUM()**

Dans le langage SQL, la fonction d'agrégation SUM() permet de calculer la somme totale d'une colonne contenant des valeurs numériques. Cette fonction ne fonction que sur des colonnes de types numériques (INT, FLOAT …) et n'additionne pas les valeurs NULL.

Syntaxe

La syntaxe pour utiliser cette fonction SQL peut être similaire à celle-ci:

```
SELECT SUM(nom_colonne) FROM table
```

Cette requête SQL permet de calculer la somme des valeurs contenu dans la colonne « nom_colonne ».
A savoir : Il est possible de filtrer les enregistrements avec la commande WHERE pour ne calculer la somme que des éléments souhaités.

Exemple

Imaginons un système qui gère des factures et enregistre chaque achat dans une base de données. Ce système utilise une table « facture » contenant une ligne pour chaque produit. La table ressemble à l'exemple ci-dessous :
Table facture :

id	facture_id	produit	prix
1	1	calculatrice	17
2	1	agrafeuse	4
3	1	ciseaux	3
4	1	agenda	15
5	2	calculatrice	17
6	2	agenda	15

Somme avec WHERE

Pour calculer le montant de la facture n°1 il est possible d'utiliser la requête SQL suivante:

```
SELECT SUM(prix) AS prix_total FROM facture  WHERE facture_id = 1
```

Résultat :

prix_total
39

Ce résultat démontre bien que tous les achats de la facture n°1 représente un montant de 39€ (somme de 17€ + 4€ + 3€ + 15€).

Somme avec GROUP BY

Pour calculer la somme de chaque facture, il est possible de grouper les lignes en se basant sur la colonne « facture_id ». Un tel résultat peut être obtenu en utilisant la requête suivante:

```
SELECT facture_id, SUM(prix) AS prix_total FROM facture GROUP BY facture_id
```

Résultat :

facture_id	prix_total
1	39
2	32

Ce résultat montre bien qu'il est possible de déterminer le prix total de chaque facture en utilisant la fonction d'agrégation SUM().

- **MERGE**

Dans le langage SQL, la commande MERGE permet d'insérer ou de mettre à jour des données dans une table. Cette commande permet d'éviter d'effectuer plusieurs requêtes pour savoir si une donnée est déjà dans la base de données et ainsi adapter l'utilisation d'une requête pour ajouter ou une autre pour modifier la donnée existante. Cette commande peut aussi s'appeler « upsert ».
Attention : bien que l'instruction a été ajoutée dans le standard SQL:2003, les différentes SGBD n'utilisent pas toutes les mêmes méthodes pour effectuer un upsert.

Syntaxe

La syntaxe standard pour effectuer un merge consiste à utiliser une requête SQL semblable à celle ci-dessous :

```
MERGE INTO table1 USING table_reference ON (conditions) WHEN MATCHED THEN  UPDATE SET table1.colonne1 = valeur1, table1.colonne2 = valeur2 DELETE WHERE conditions2  WHEN NOT MATCHED THEN  INSERT (colonnes1, colonne3) VALUES (valeur1, valeur3)
```

Voici les explications détaillées de cette requête :

1. **MERGE INTO** permet de sélectionner la table à modifier
2. **USING** et **ON** permet de lister les données sources et la condition de correspondance
3. **WHEN MATCHED** permet de définir la condition de mise à jour lorsque la condition est vérifiée
4. **WHEN NOT MATCHED** permet de définir la condition d'insertion lorsque la condition n'est pas vérifiée.

Compatibilité

Les systèmes de gestion de bases de données peuvent implémenter cette fonctionnalité soit de façon standard, en utilisant une commande synonyme ou en utilisant une syntaxe non standard.

- **Syntaxe standard :** SQL Server, Oracle, DB2, Teradata et EXASOL
- **Utilisation du terme UPSERT :** Microsoft SQL Azure et MongoDB
- **Utilisation non standard :** MySQL, SQLite, Firebird, IBM DB2 et Microsoft SQL

- **DROP DATABASE**

En SQL, la commande DROP DATABASE permet de supprimer totalement une base de données et tout ce qu'elle contient. Cette commande est à utiliser avec beaucoup d'attention car elle permet de supprimer tout ce qui est inclus dans une base: les tables, les données, les index …

Syntaxe

Pour supprimer la base de données « ma_base », la requête est la suivante :

```
DROP DATABASE ma_base;
```

Attention : cela va supprimer toutes les tables et toutes les données de cette base. Si vous n'êtes pas sûr de ce que vous faites, n'hésitez pas à effectuer une sauvegarde de la base avant de supprimer.

Ne pas afficher d'erreur si la base n'existe pas

Par défaut, si le nom de base utilisé n'existe pas, la requête retournera une erreur. Pour éviter d'obtenir cette erreur si vous n'êtes pas sûr du nom, il est possible d'utiliser l'option IF EXISTS. La syntaxe sera alors la suivante:

```
DROP DATABASE IF EXISTS ma_base;
```

- **ORDER BY**

  La commande 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.

Syntaxe

Une requête où l'on souhaite filtrer l'ordre des résultats utilise la commande ORDER BY de la sorte :

```
SELECT colonne1, colonne2 FROM table ORDER BY colonne1
```

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. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule. Une requête plus élaborée ressemblerait à cela :

```
SELECT colonne1, colonne2, colonne3 FROM table
ORDER BY colonne1 DESC, colonne2 A
```

A noter : il n'est pas obligé d'utiliser le suffixe « ASC » sachant que les résultats sont toujours classés par ordre ascendant par défaut. Toutefois, c'est plus pratique pour mieux s'y retrouver, surtout si on a oublié l'ordre par défaut.

Exemple

Pour l'ensemble de nos exemples, nous allons prendre une base « utilisateur » de test, qui contient les données suivantes :

Id	nom	prenom	date_inscription	tarif
1	Dupont	Pierre	2018-07-12	123
2	Gauriau	Marie-Hélène	2018-07-12	456
3	Jaadar	Houda	2013-07-02	123
4	Diarra	Samba	2018-07-12	456
5	Martin	Jean	2018-11-12	789

Pour récupérer la liste de ces utilisateurs par ordre alphabétique du nom de famille, il est possible d'utiliser la requête suivante :

```
SELECT * FROM utilisateur ORDER BY nom
```

Résultat :

Id	nom	prenom	date_inscription	tarif
4	Diarra	Samba	2018-07-12	456
1	Dupont	Pierre	2018-07-12	123
2	Gauriau	Marie-Hélène	2013-07-02	456
3	Jaadar	Houda	2018-07-12	123
5	Martin	Jean	2018-11-12	789

En utilisant deux méthodes de tri, il est possible de retourner les utilisateurs par ordre alphabétique ET pour ceux qui ont le même nom de famille, les trier par ordre décroissant d'inscription. La requête serait alors la suivante :

```
SELECT * FROM utilisateur ORDER BY nom, date_inscription DESC
```

Résultat :

Id	nom	prenom	date_inscription	tarif
4	Diarra	Samba	2018-07-12	456
1	Dupont	Pierre	2018-07-12	123
2	Martin	Jean	2013-07-1	789

			2	
3	Jaadar	Houda	2018-07-12	123
2	Gauriau	Marie-Hélène	2013-07-02	456