

---

---

# **COURS SQL**

## **BTS SIO 1e année**

par Mickaël LE BRAS

---



# Introduction

**Les bases de ce que nous allons voir dans ce cours**

- **Le SQL, c'est quoi ?**  
Introduction à la notion de base de données, et au langage SQL.
- **L'installation**  
Installation des outils qui vont nous permettre de travailler avec le SQL pour le web
- **Projection**  
Visualiser les informations de notre base de données
- **Requêtes**  
Manipuler les informations

# Le SQL, c'est quoi?

- Langage créé dans les années 60
- Permet de dialoguer avec des banques de données
- N'est pas un langage de programmation (Non procédural)
- On utilise le SQL via des Commandes
- Le SQL permet de manipuler des données (récupérer, créer, modifier, supprimer)
- Le SQL permet de manipuler une base de données ( création d'une Table, d'une colonne...)



# Premières notions

Le SQL se structure de la manière suivante :

- **Base de Données** : ensemble d'informations structurées  
exemple : Une base de données d'une école, d'un entrepôt...
- **Table** : ensemble de données organisées sous forme d'un tableau  
exemple : Liste des élèves, liste des produits...
- **Colonne** : catégories d'information  
exemple : Noms des élèves, prix des produits...
- **Ligne** : enregistrements de données pour une entité  
exemple : Martin pour le nom d'un élève et Jean pour son prénom

# Les outils

Le SQL fonctionne à l'aide d'un SGBD ou d'un SGBDR :

- **SGBD** : Système de Gestion de Bases de Données

C'est un logiciel système servant à stocker, à manipuler ou gérer, et à partager des données dans une base de données, en garantissant la qualité, la pérennité et la confidentialité des informations, tout en cachant la complexité des opérations.

- **SGBDR** : Système de Gestion de Bases de Données Relationnelles

C'est un logiciel système qui reprend tous les principes du SGBD en lui ajoutant la notion de Relations. La relation va permettre de faire correspondre plusieurs données entre elles (par exemple: le nom, le prénom et l'âge d'un élève sont des données liées parce qu'elles correspondent toutes à un seul et même élève).

# L'installation des outils

Étant donné que nous avons entamé une formation en développement web, nous allons installer **MySQL**, un **SGBDR** très répandu pour ce type de programmation.

Nous pourrions installer directement **MySQL** mais nous allons faire d'une pierre deux coups en installant un outil qui s'appelle **WAMP** ou **MAMP**.

**WAMP** ou **MAMP** va nous permettre de :

- Posséder un serveur **WEB** en local sur notre ordinateur.

Ce qui nous sera utile quand nous souhaiterons mettre en relation nos sites web avec des bases de données, ou avec un langage de programmation Serveur.

- Installer **PHP**

Que nous étudierons plus tard dans l'année

- Installer **MySQL**

WINDOWS	MAC
<a href="https://www.wampserver.com/">https://www.wampserver.com/</a>	<a href="https://www.mamp.info/">https://www.mamp.info/</a>

# L'accès à MySQL

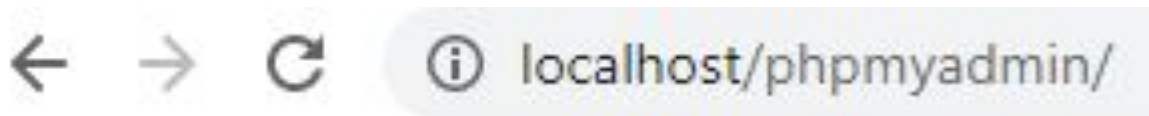
Une fois l'installation de WAMP ou de MAMP terminée, exécutez le logiciel.

WAMP ou MAMP va alors démarrer 3 services :

- Un **serveur WEB Apache** qui permettra de faire de la programmation en PHP
- Un **SGBDR MariaDB**
- Un **SGBDR MySQL**

MariaDB et MySQL fonctionnent de la même manière.

Pour accéder à vos bases de données, saisissez l'url suivante dans votre navigateur:



# Connexion

1. Saisissez votre nom d'utilisateur (Username) :  
"root"
2. Si vous n'avez pas spécifié de mot de passe à l'installation, laissez le champ (Password) vide. Sinon, saisissez votre mot de passe.
3. Cliquez sur "Go" !
4. Bienvenue sur votre tableau de bord phpMyAdmin !



The screenshot shows the phpMyAdmin login page. At the top, there is a logo with a sailboat and the text "phpMyAdmin". Below the logo, it says "Welcome to phpMyAdmin". There are two main sections: a "Language" section with a dropdown menu set to "English", and a "Log in" section. The "Log in" section contains three fields: "Username:" with the value "root", "Password:" which is empty, and "Server Choice:" with a dropdown menu set to "MariaDB". A "Go" button is located at the bottom right of the "Log in" section.



# Vos bases de données

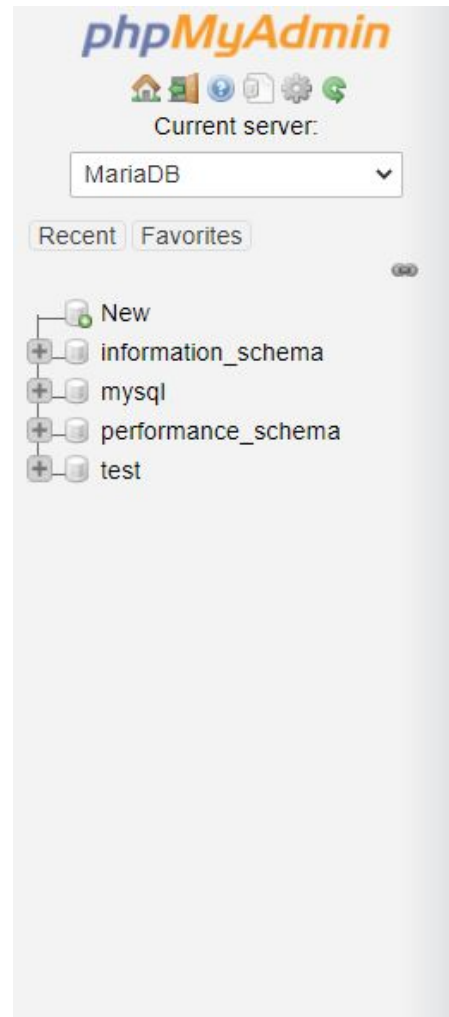
L'ensemble de vos bases de données vont se présenter sur le menu de gauche.

Pour en créer une nouvelle, vous pourrez cliquer sur le bouton "New". Nous le ferons juste après.

4 bases de données sont présentes de base lors de l'installation de **WAMP** ou **MAMP**, et nous n'y toucherons pas.

De même, pour le moment, nous allons laisser tout le reste de la page de côté.

Nous allons pouvoir revenir au **SQL** !



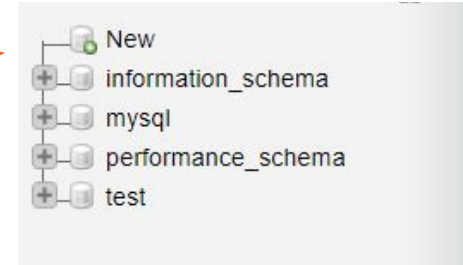
# Notre première base de données

Cliquez maintenant sur le bouton “New” pour créer une base de données.

Saisissez “ecole” dans le champ “Database Name” (**SANS ACCENT**)

Si ce n’est pas déjà le cas, sélectionnez “utf8\_general\_ci” dans la liste déroulante.

Puis, cliquez sur “Create”



Databases

Create database

Database name


utf8\_general\_ci

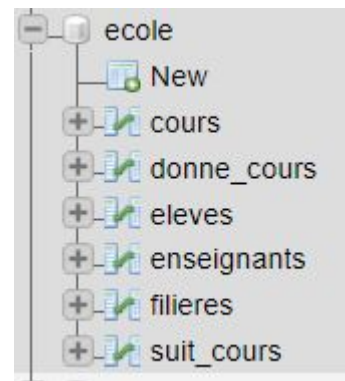
Create

# Importez le contenu

Nous avons une base de données “**ecole**”, mais elle est vide.

Nous verrons plus tard comment créer des tables, des colonnes et des lignes. Pour le moment on va importer des données directement depuis un fichier.

1. Téléchargez le fichier **ecole.sql**
2. Sur Phpmyadmin, cliquez sur la base de données “**ecole**” dans la liste de gauche
3. Dans le menu du haut, cliquez sur  Import
4. Importez le fichier **ecole.sql**
5. Laissez les options par défaut
6. Cliquez sur “Go”
7. Constatez le changement dans la liste des bases de données !



# Questions

1. Combien de tables contient la base de données “ecole” ?
2. Combien de colonnes contient “eleves” ?
3. A quelle table appartient la colonne “Titre” ?
4. Quel est la valeur de la colonne “Resume” du cours ayant comme “Intitule” la valeur “Développement” ?
5. Quel est la valeur de la colonne “Prenom” quand la colonne “MatriculeEnseignant” possède la valeur 4 ?
6. D’après vous, à quoi servent les tables :
  - a. “donne\_cours”
  - b. “suit\_cours”
7. Quel est le point commun entre les colonnes “MatriculeEnseignant” de la table “enseignants” et “MatriculeEleve” de la table “eleve” ?

# La Projection

Une projection est une instruction permettant de sélectionner un ensemble de colonnes dans une table :

Par exemple la projection de la table “Cours” est :

				NumCours	Intitule	Resume
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Mathématiques	Les mathématiques (ou la mathématique) sont un ens...
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Français	Le français est une langue indo-européenne de la f...
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Anglais	L'anglais est une langue indo-européenne germaniqu...
<input type="checkbox"/>	 Edit	 Copy	 Delete	4	Développement	Création de programmes, sites web et autres applic...
<input type="checkbox"/>	 Edit	 Copy	 Delete	5	Réseau	Un réseau informatique (en anglais, data communica...
<input type="checkbox"/>	 Edit	 Copy	 Delete	6	Algorithmie	L'algorithmique est l'étude et la production de rè...
<input type="checkbox"/>	 Edit	 Copy	 Delete	7	Économie	L'économie rassemble les activités humaines tourné...

# La Sélection

Une sélection est une requête SQL qui permet de récupérer seulement un panel de valeurs de notre choix.

Elle se fait à l'aide de l'instruction SELECT

Par exemple, pour récupérer toutes les valeurs de la table Eleves, on fera :

**SELECT \* FROM eleves**

Le signe \* indique que l'on sélectionne toutes les valeurs

Pour lancer une requête SQL sur Phpmyadmin il faut cliquer sur



Ensuite, saisissez la requête ci-dessus (en orange) et exécutez la

# La Sélection (suite)

On peut également choisir de sélectionner uniquement les prénoms et les noms des élèves, pour celà, on remplacera le signe \* par les colonnes que l'on veut récupérer en les séparant par des virgules.

```
SELECT Prenom, Nom FROM eleves
```

Ici, nous allons récupérer les colonnes **Prenom** et **Nom** de la table **eleves**

Nous pouvons également renommer une colonne grâce au mot clé **AS** qui permet la création d'un ALIAS.

```
SELECT Prenom AS LePrenom FROM eleves
```

Ici, nous allons récupérer la colonne **Prenom**, mais dans la projection, elle s'intitule **LePrenom**

# Le DISTINCT

Lors d'une sélection, il se peut que deux valeurs soient identiques. Si l'on ne souhaite afficher qu'une fois chaque valeurs, il faudra utiliser la commande **DISTINCT**.

```
SELECT DISTINCT Titre FROM enseignants
```

Que nous renvoi cette requête ?



# Les fonction d'agrégations

Les fonctions d'agrégation dans le langage SQL permettent d'effectuer des opérations statistiques sur un ensemble d'enregistrement.

On peut récupérer :

- Une Moyenne
- Un Compteur
- Une Valeur maximale
- Une Valeur minimale
- Une Somme

# Les fonction d'agrégations:

## La Moyenne

Pour récupérer une moyenne, on utilisera la fonction d'agrégation **AVG()**

Par exemple, rédigez cette requête SQL :

```
SELECT AVG(Note) FROM notes
```

Que nous renvoi cette requête ?

# Les fonction d'agrégations:

## Le Compteur

Pour récupérer un compteur, on utilisera la fonction d'agrégation **COUNT()**

Par exemple, rédigez cette requête SQL :

```
SELECT COUNT(Note) FROM notes
```

Que nous renvoi cette requête ?

# Les fonction d'agrégations: La Valeur Maximale

Pour récupérer la valeur maximale, on utilisera la fonction d'agrégation **MAX()**

Par exemple, rédigez cette requête SQL :

```
SELECT MAX(Note) FROM notes
```

Que nous renvoi cette requête ?

# Les fonction d'agrégations:

## La Valeur Minimale

Pour récupérer la valeur minimale, on utilisera la fonction d'agrégation **MIN()**

Par exemple, rédigez cette requête SQL :

```
SELECT MIN(Note) FROM notes
```

Que nous renvoi cette requête ?

# Les fonction d'agrégations:

## La Somme

Pour récupérer une somme, on utilisera la fonction d'agrégation **SUM()**

Par exemple, rédigez cette requête SQL :

```
SELECT SUM(Note) FROM notes
```

Que nous renvoi cette requête ?

# Le GROUP BY

Il peut être utile de regrouper certaines données quand nous utilisons une fonction d'agrégation.

Pour cela, le mot clé à utiliser est GROUP BY et il se place à la fin de la requête.

Par exemple, on peut exécuter cette requête :

```
SELECT NumCours, AVG(Note) FROM notes GROUP BY NumCours
```

Que nous renvoi cette requête ?

# Le ORDER BY

On peut également trier les résultats par ordre croissant ou décroissant grâce à la commande ORDER BY.

Par défaut, les résultats sont classés par ordre Ascendant, mais on peut spécifier que l'on attend un ordre descendant.

Ordre Ascendant : **ASC**

Ordre Descendant : **DESC**

```
SELECT NumCours, AVG(Note)
FROM notes
GROUP BY NumCours
ORDER BY NumCours DESC
```

Que nous renvoi cette requête ?



# Exercices

1. Rédigez la requête qui affiche le numéro de la filière et le nombre d'élèves de cette filière (avec un alias).
2. Rédigez la requête qui affiche le numéro du cours et la moyenne de toutes les notes de ce cours (avec un alias).
3. Rédigez la requête qui affiche le numéro du cours, la note minimale (avec un alias) et la note maximale (avec un alias) de ce cours.
4. Rédigez la requête qui affiche tous les titres des enseignants et le nombre d'enseignants possédant ce titre (avec un alias).
5. Rédigez la requête qui affiche les numéros des filières et le nombre de matières enseignées dans cette filière (avec un alias)
6. Rédigez la requête qui permet pour chaque Numéro de matière et pour chaque Matricule d'élève, d'afficher la note Maximale (avec un alias)

# Le 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.

La commande **WHERE** se place toujours après le **FROM** et avant les éventuels **GROUP BY** et **ORDER BY**.

On utilise des simples quotes pour insérer une chaîne de caractères dans une requête SQL.

```
SELECT * FROM enseignants WHERE Titre = 'Docteur'
```

Que renvoi cette requête ?

# Le WHERE : AND et OR

Une requête SQL peut être restreinte à l'aide de la condition **WHERE**. Les opérateurs logiques **AND** et **OR** peuvent être utilisées au sein de la commande **WHERE** pour combiner des conditions.

```
SELECT * FROM eleves WHERE NumFiliere = 1 AND Prenom = 'Riley'
```

Que renvoi cette requête ?

```
SELECT * FROM enseignants WHERE Titre = 'Docteur' OR Titre = 'Monsieur'
```

Que renvoi cette requête ?

# Le WHERE : Les opérateurs

On peut effectuer des comparaisons différentes du “=” en SQL. En voici la liste:

= : Égale

<= : Inférieur ou égale à

<> : Pas Égale

IN : Plusieurs valeurs possibles

!= : Pas Égale

BETWEEN : Interval de valeurs

> : Supérieur à

LIKE : Contient une chaîne

< : Inférieur à

IS NULL : Valeur est nulle

>= : Supérieur ou égale à

IS NOT NULL : Valeur n'est pas nulle

# L'opérateur IN

L'opérateur logique **IN** dans SQL s'utilise avec la commande **WHERE** pour vérifier si une colonne est égale à une des valeurs comprise dans une liste de valeurs déterminés.

```
SELECT Nom  
FROM eleves  
WHERE Prenom IN ('Charlie', 'Phoenix')
```

Que nous renvoi cette requête ?

# L'opérateur BETWEEN

L'opérateur **BETWEEN** est utilisé dans une requête SQL pour sélectionner un interval de données dans une requête utilisant **WHERE**.

```
SELECT Prenom, Nom  
FROM eleves  
WHERE Naissance BETWEEN '2000-01-01' AND '2000-12-31'
```

Que nous renvoi cette requête ?

# L'opérateur 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. On se sert du caractère **%** pour remplacer tout le reste de la chaîne.

Les modèles de recherche les plus courants sont :

- La valeur commence par une chaîne de caractère spécifique  
**LIKE 'Bonjour%'** : La valeur commence par "Bonjour"
- La valeur finit par une chaîne de caractère spécifique  
**LIKE '%Au revoir'** : La valeur termine par "Au revoir"
- La valeur contient une chaîne de caractère  
**LIKE '%a%'** : La valeur comprend la lettre "a"
- La valeur commence et finit par une chaîne de caractère  
**LIKE 'a%z'** : La valeur commence par la lettre "a" et finit par la lettre "z"

# Exercices

1. Rédigez la requête qui affiche les matricules et les titres des professeurs qui ont un titre non vide.
2. Rédigez la requête qui affiche le numéro du cours et la moyenne de toutes les notes de ce cours (avec un alias) pour les élèves nés entre le 01 janvier 1999 et le 31 décembre 2001.  
(AJOUTEZ LA LIGNE : **JOIN** **eleves** **USING** (**MatriculeEleve**) juste après le **FROM** **notes**)
3. Rédigez la requête qui affiche le Prénom et le Nom de tous les élèves dont le prénom ou le nom commence par la lettre S ou finit par la lettre N .
4. Rédigez la requête qui affiche le Matricule de l'élève, le numéro du cours et sa moyenne par cours quand l'élève est né avant l'année 2000.  
(AJOUTEZ LA LIGNE : **JOIN** **eleves** **USING** (**MatriculeEleve**) juste après le **FROM** **notes**)
5. Rédigez la requête qui affiche le nombre d'enseignants qui n'ont pas de titre, mais qui sont responsables d'une filière.
6. Rédigez la requête qui permet d'afficher la filière et le nombre d'élèves compris dedans pour ceux qui sont nés en 1999 et ont un prénom qui commence par la lettre "L".



# Les Clés Primaires

- **La Clé Primaire** : C'est une colonne qui permet d'identifier de manière unique un enregistrement dans une table. Il peut y avoir une combinaison de clés primaires dans des tables d'associations. En général, on lui précise l'attribut "**AUTO\_INCREMENT**" qui permet d'ajouter +1 à chaque nouvelle ligne dans la table. Sur Phpmyadmin, on voit la clé primaire grâce à la petite clé dorée.

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
MatriculeEnseignant 	int(11)			No	None		AUTO_INCREMENT

## Questions :

1. Quelle est la clé primaire de la table eleves ?
2. Quelle est la clé primaire de la table cours?
3. De la table notes?
4. Quelles sont les clés primaires de la table suit\_cours ?

# La clause LIMIT / OFFSET

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).

Exemple 1 :

```
SELECT MatriculeEleve, Nom  
FROM eleves  
LIMIT 5
```

Exemple 2:

```
SELECT MatriculeEleve, Nom  
FROM eleves  
LIMIT 5 OFFSET 5
```

# 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.

Une seule ligne :

```
INSERT INTO eleves (Nom, Prenom, Naissance, NumFiliere)  
VALUES ("Riner", "Teddy", "2000-08-09", 1)
```

Plusieurs lignes:

```
INSERT INTO eleves(Nom, Prenom, Naissance, NumFiliere)  
VALUES ("Zidane", "Zinedine", "1998-07-12", 1),  
("Deschamps", "Didier", "1998-07-12", 2)
```

# 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.

**UPDATE** eleves

**SET** Nom = "Reeves", Prenom = "Keanu"

**WHERE** MatriculeEleve = 5

# ON DUPLICATE KEY UPDATE

L'instruction **ON DUPLICATE KEY UPDATE** est une fonctionnalité de MySQL qui permet de mettre à jour des données lorsqu'un enregistrement existe déjà dans une table. Cela permet d'avoir qu'une seule requête SQL pour effectuer selon la convenance un INSERT ou un UPDATE.

On devra saisir la clé primaire car MySQL se basera sur cette colonne pour vérifier si le champ existe déjà.

```
INSERT INTO eleves (MatriculeEleve, Nom, Prenom, Naissance, NumFiliere)  
VALUES (1, "Coursevent", "Sylvanas", "1998-11-27", 1)  
ON DUPLICATE KEY UPDATE Nom = "Coursevent", Prenom = "Sylvanas", Naissance =  
"1998-11-27", NumFiliere = 1
```

# 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. Sans condition, toutes les lignes de la table seront supprimées.

Exemple 1 :

```
DELETE FROM eleves  
WHERE MatriculeEleve = 5
```

Exemple 2 :

```
DELETE FROM eleves  
WHERE Naissance < "1997-01-01"
```

# TRUNCATE

En SQL, la commande **TRUNCATE** permet de supprimer toutes les données d'une table sans supprimer la table en elle-même. En d'autres mots, cela permet de purger la table.

## TRUNCATE TABLE filieres

La différence avec la commande **DELETE** est :

- **TRUNCATE** réinitialise les valeurs de l'auto incrémentation s'il y en a un  
Donc la prochaine insertion prendra 1 en clé primaire.
- **DELETE** ne réinitialise pas les valeurs de l'auto incrémentation.  
Donc la prochaine insertion continuera à partir de la précédente valeur maximale (si avant le **DELETE**, la clé primaire maximale était à 20, après le **DELETE**, on insèrera la ligne 21).

# 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.

```
CREATE DATABASE ma_base_de_test
```



# 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 ...

**DROP DATABASE** ma\_base\_de\_test

# 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...).

**CREATE TABLE** salles

```
(  
    NumSalle INT,  
    Batiment VARCHAR(1)  
)
```

Par défaut, les champs ainsi créés seront “nullables”, c’est à dire qu’ils accepteront une valeur nulle mais vous pouvez spécifier qu’ils sont obligatoires en rajoutant **NOT NULL** : **NumSalle INT NOT NULL**,

# 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.

**DROP TABLE** salles

# PRIMARY KEY et AUTO\_INCREMENT

Dans le langage SQL la “**PRIMARY KEY**” autrement dit la clé primaire, permet d’identifier chaque enregistrement dans une table de base de données. Chaque enregistrement de cette clé primaire doit être UNIQUE et ne doit pas contenir de valeur NULL.

La commande **AUTO\_INCREMENT** est utilisée afin de spécifier qu’une colonne numérique avec une clé primaire (**PRIMARY KEY**) sera incrémentée automatiquement à chaque ajout d’enregistrement dans celle-ci.

L’usage courant de **PRIMARY KEY** peut être effectué lors de la création d’une table à l’aide de la syntaxe suivante :

```
CREATE TABLE salles  
(  
    NumSalle INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    Batiment VARCHAR(1)  
)
```

# Multiples clés primaires

Lorsque l'on crée une table dite d' "Association" (une table qui a deux clés primaires et qui permet de relier les informations de deux autres tables entre elles), on a besoin d'utiliser une syntaxe particulière pour définir nos deux clés primaires :

```
CREATE TABLE suit_cours
(  
    NumFiliere INT,  
    NumCours INT,  
    PRIMARY KEY (NumFiliere, NumCours)  
)
```

# ALTER TABLE

La commande **ALTER TABLE** en SQL permet de modifier une table existante. Idéal pour ajouter une colonne, supprimer une colonne ou modifier une colonne existante, par exemple pour changer le type. Le mot clé est **ADD** pour ajouter une colonne, **MODIFY** pour modifier seulement le type d'une colonne, **DROP COLUMN** pour supprimer une colonne et **CHANGE** pour renommer une colonne et changer son type. Dans un **ALTER TABLE**, on sépare les instructions par une virgule.

```
ALTER TABLE salles  
ADD Capacite INT,  
MODIFY Batiment INT
```

```
ALTER TABLE salles  
DROP COLUMN Capacite,  
CHANGE Batiment CodeBatiment VARCHAR(1)
```

# Exercice entrepôt

Notez toutes vos requêtes dans un document texte.

Info importante: Convention d'écriture  
table(cle primaire (type), colonne (type))

1. Créez une nouvelle base de données appelée **entrepot**
2. Dans cette base de données créez 3 tables (n'oubliez pas l'auto increment):
  - a. **categories** (id\_categorie (int), titre (varchar(100))
  - b. **produits** (id\_produit (int), titre (varchar(100)), marque (varchar(100)), modele (varchar(100)), emplacement (int), id\_allée (int), id\_categorie (int))
  - c. **allée** (id\_allée (int), code\_allée (varchar(1))
3. Écrire une requête qui modifie la colonne "titre" de la table produits en "designation", et qui change son nombre maximal de caractères de 100 à 150.
4. Écrire une requête qui permet d'insérer **deux** catégories de produits de votre choix, et une requête qui insère **deux** allées (A et B).
5. Écrire une requête qui permet d'insérer **cinq** produits de votre choix
6. Écrire une requête qui modifie la catégorie ayant l'id 1, une autre qui modifie le produit ayant l'id 5
7. Écrire une requête qui permet de créer une table **fournisseurs** (id\_fournisseur (int), nom (varchar(100)) (n'oubliez pas l'auto increment).
8. Écrire une requête qui permet d'insérer trois fournisseurs de votre choix.
9. Écrire une requête qui permet de modifier la table produits pour ajouter l'id\_fournisseur
10. Écrire une requête par produit qui permet de le modifier pour lui assigner un id\_fournisseur
11. Écrire la requête qui permet de supprimer le produit ayant l'id 2
12. Écrire la requête qui permet de vider sans réinitialiser la table allée, écrire la requête qui permet de créer les allées C et D, écrire la requête qui modifie les produits et qui remplace les allées A par les allées C, et les allées B par les allées D
13. Écrire sans l'exécuter la requête qui permettrait de supprimer la table allée, et celle qui permettrait de supprimer les données sans la réinitialiser
14. Écrire la requête qui permet de récupérer les 2 premiers produits de la 2e catégorie, puis la requête qui permet de récupérer les 2 suivants.

# Exercice animalerie

Notez toutes vos requêtes dans un document texte.

**Info importante: Convention d'écriture**  
table(cle primaire (type), colonne (type))

1. Créez une nouvelle base de données appelée **animalerie**
2. Dans cette base de données créez 3 tables (n'oubliez pas l'auto increment, et tous les champs seront obligatoires dans toutes les tables):
  - a. animaux (id\_animal (int), race(varchar(100)), nombre\_pattes(int), id\_espece(int), prix(decimal))
  - b. especes(id\_espece (int), nom(varchar(100)))
  - c. clients(id\_client (int), nom (varchar(100)), prenom (varchar(100)))
3. Écrire une requête qui modifie la table clients et qui ajoute deux colonnes : adresse\_email qui sera une chaîne de caractères d'une longueur de 100 et date\_naissance qui sera au format date. Ces champs seront aussi obligatoires.
4. Écrire une requête qui permet d'insérer **trois** especes (chiens, chats et lapins), et **deux** animaux par espèces de votre choix.
5. Écrire une requête qui permet d'insérer **cinq** clients de votre choix
6. Écrire une requête qui modifie le client ayant l'id 5, une autre qui modifie l'animal ayant l'id 5
7. Écrire une requête qui permet de créer une table rayons(id\_rayon (int), titre(varchar(100))) (n'oubliez pas l'auto increment).
8. Écrire une requête qui permet d'insérer **deux** rayons : animaux de compagnie et rongeurs.
9. Écrire une requête qui permet de modifier la table especes pour ajouter l'id du rayon dans lequel elle se situe, et écrire la requête qui permet d'associer chaque espèce à son rayon
10. Créer une nouvelle table achats(id\_animal(int), id\_client(int))
11. Écrire la requête qui permet de faire en sorte que le client 3 ait acheté l'animal 1, et que le client 4 ait acheté l'animal 6
12. Écrire la requête qui permet de récupérer le nombre d'animaux par especes (sous cette forme : [id de l'espece], [nombre d'animaux])
13. Écrire sans l'exécuter la requête qui permettrait de supprimer la table achats, et celle qui permettrait de modifier chaque id\_client de la table achat pour rajouter 1 (par exemple, l'id\_client 1 vaudrait maintenant 2).
14. Écrire la requête qui permet de récupérer le premier client ayant acheté un animal, puis la requête qui permet de récupérer uniquement le deuxième client ayant acheté un animal (en utilisant le offset)



# Exercice films

**Info importante: Convention d'écriture**  
table(cle primaire (type), colonne (type))

**Schéma (tous les champs sont obligatoires) :**

Film (idFilm (int), titre (varchar(100)), année int(4), genre (varchar(100)), résumé (text), idRealisateur (int), codePays (int))

Pays (code (int), nom (varchar(100)), langue (varchar(100)))

Artiste (idArtiste (int), nom (varchar(100)), prenom (varchar(100)), anneeNaiss (int(4))

Rôle (idFilm (int), idActeur (int), nomRole (varchar(100)))

Internaute (email (varchar(100)), nom (varchar(100)), prénom (varchar(100)), region (varchar(100)))

Notation (email (varchar(100)), idFilm (int), note (int(2)))

**Écrire les requêtes qui permettent :**

1. De créer toutes les tables (si MySQL sort une erreur pour la colonne code, mettez "code")
2. D'insérer 4 artistes nés en 1930, 1940, 1960 et 1970 et une autre qui permet de récupérer les Nom et année de naissance des artistes nés avant 1950.
3. De récupérer tous les artistes classés par leur année de naissance de la plus grande à la plus petite pour ceux nés avant l'année 1950 et une autre qui fait la même chose mais uniquement pour ceux nés après l'année 1950
4. D'insérer un film, une autre qui permet d'insérer un internaute, et une autre requête qui permet d'ajouter une notation à ce film pour cet internaute.
5. D'insérer deux pays, dont un ayant comme code celui que vous avez déjà saisi pour le film
6. D'insérer un nouveau film et une autre requête permettant de faire en sorte que les 4 artistes que vous avez déjà ajouté aient un rôle dans ce film (recherchez la table qui correspond à cela)
7. De modifier la structure de la table Pays pour que le code devienne idPays, une autre permettant de modifier la structure de la table Film pour que le codePays devienne idPays, une autre permettant d'ajouter un champ dans la table Internaute pour connaître son pays.
8. De saisir une note pour le 2e film, et une autre requête qui permet de récupérer le nombre de films ayant une note inférieure à 5, et le nombre de films ayant une note supérieure ou égale à 5

# Les contraintes

Les contraintes sont les règles appliquées aux colonnes de données d'une table. Celles-ci sont utilisées pour limiter le type de données pouvant aller dans une table. Cela garantit l'exactitude et la fiabilité des données de la base de données.

Les contraintes les plus communes sont :

- NOT NULL
- PRIMARY KEY
- UNIQUE

La contrainte **UNIQUE** permet de spécifier à MySQL que la colonne que l'on crée ou que l'on modifie devra avoir une valeur qui devra être unique dans la table (donc aucun doublon ne sera accepté).

# Exercice bibliothèque

Info importante: Convention d'écriture  
table(cle primaire (type), colonne (type))

Schéma (tous les champs sont obligatoires) :

Livre (idLivre(int), titre (varchar(100)), annee int(4), resume (text), idGenre(int), idAuteur (int))

Auteur(idAuteur(int), nom (varchar(100)), prenom (varchar(100)), anneeNaiss (int(4))

Genre (idGenre(int), nomGenre (varchar(100)))

Client (idClient(int), nom (varchar(100)), prenom (varchar(100)))

Emprunt(idClient(int), idLivre(int), date\_emprunt(DATETIME), date\_retour(DATETIME))

Écrire les requêtes qui permettent :

1. De créer la base de données et de créer toutes les tables
2. De créer 2 genres, 2 auteurs et 4 livres. (En 3 requêtes différentes)
3. De récupérer tous les auteurs possédant la lettre e dans leur prénom ou leur nom, et dont l'année de naissance se situe entre 1980 et 2000, par ordre alphabétique inverse du nom de famille
4. De modifier la table Client et d'y ajouter un champ email qui sera une chaîne de caractères de maximum 100 caractères et qui devra obligatoirement être unique dans la table.
5. De créer 2 clients de votre choix, puis de faire en sorte que les deux clients aient empruntés un livre. (table emprunt)
6. De récupérer le nombre de clients différents ayant empruntés un livre, et de récupérer le nombre de livres différents ayant été empruntés.
7. De supprimer toutes les données de la table emprunt, en la réinitialisant et de supprimer toutes les données de la table genre sans la réinitialiser.
8. De récupérer tous les livres dont le titre possède la lettre L en premier caractère, contient la lettre E et qui termine par la lettre X dont l'année de parution se situe entre 1930 et 1960. Classés par titre alphabétiquement décroissant.
9. De récupérer le nombre de livres par années de parutions par ordre d'année croissante.
10. De récupérer le nombre d'auteurs ayant la même année de naissance, par années décroissantes
11. D'ajouter un nouveau client ayant la même adresse email qu'un client déjà existant. Que se passe-t-il ?

# Les jointures

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.

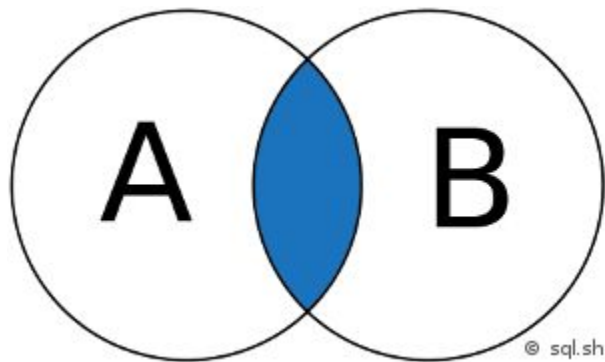
En général, les jointures consistent à associer des lignes de 2 tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table.

Imaginons qu'une base de 2 données possède une table "utilisateur" et une autre table "adresse" qui contient les adresses de ces utilisateurs. Avec une jointure, il est possible d'obtenir les données de l'utilisateur et de son adresse en une seule requête.

Il existe plusieurs types de jointures.

# INNER JOIN

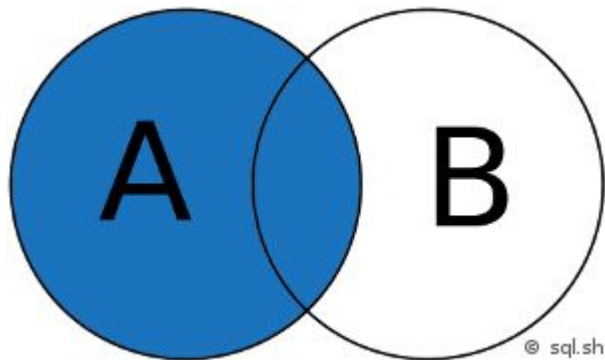
La commande **INNER JOIN**, est un type de jointures très communes pour lier plusieurs tables entre-elles. Cette commande retourne les enregistrements lorsqu'il y a au moins une ligne dans chaque colonne qui correspond à la condition.



```
SELECT *  
FROM eleves  
INNER JOIN notes ON notes.MatriculeEleve = eleves.MatriculeEleve
```

# LEFT JOIN

La commande **LEFT JOIN** est un type de jointure entre 2 tables. Cela permet de lister tous les résultats de la table de gauche (left = gauche) même s'il n'y a pas de correspondance dans la deuxième tables.



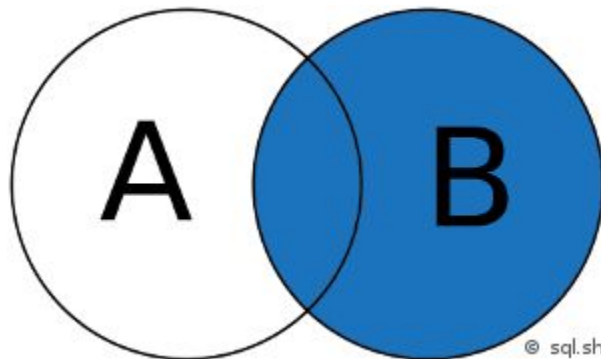
**SELECT \***

**FROM enseignants**

**LEFT JOIN filieres ON enseignants.NumFiliere = filieres.NumFiliere**

# RIGHT JOIN

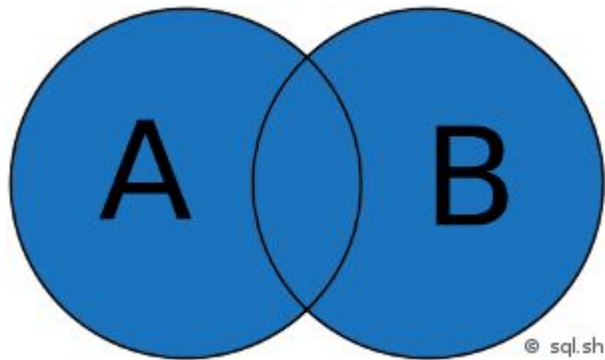
La commande **RIGHT JOIN** est un type de jointure entre 2 tables qui permet de retourner tous les enregistrements de la table de droite (right = droite) même s'il n'y a pas de correspondance avec la table de gauche.



```
SELECT *  
FROM filieres  
RIGHT JOIN enseignants ON enseignants.NumFiliere = filieres.NumFiliere
```

# FULL JOIN

La commande **FULL JOIN** permet de faire une jointure entre 2 tables. L'utilisation de cette commande permet de combiner les résultats des 2 tables, les associer entre eux grâce à une condition et remplir avec des valeurs NULL si la condition n'est pas respectée. MySQL ne supporte pas cette commande, mais il peut être utile de la connaître.



```
SELECT *  
FROM filieres  
FULL JOIN enseignants ON enseignants.NumFiliere = filieres.NumFiliere
```



# SELF JOIN

Un **SELF JOIN** correspond à une jointure d'une table avec elle-même. Ce type de requête n'est pas si commun mais très pratique dans le cas où une table lie des informations avec des enregistrements de la même table.

Il s'agit donc simplement de faire une jointure d'une table avec elle-même (de type LEFT, RIGHT, INNER, FULL ...).

Par exemple, imaginons un projet dans lequel nous aurons des catégories et des sous-catégories. Au lieu de créer deux tables (categories et sous\_categories), on peut n'en créer qu'une seule et ajouter l'id de la catégorie parente dans la table de cette façon :

`categories (id_categorie (int), intitule (varchar(100)), id_parent (int, null))`

Donc la requête pour récupérer les intitulés des sous catégories qui ont un parent, et l'intitulé des parents sera :

```
SELECT sous_cat.intitule as "Intitulé sous catégorie", cat.intitule as "Intitulé catégorie parente"  
FROM categories sous_cat  
INNER JOIN categories cat ON sous_cat.id_parent = cat.id_categorie
```

# Exercice sur les jointures

Effectuez cet exercice sur la base "ecole"

Écrire les requêtes qui permettent :

1. De lister pour chaque élève son Nom, son Prénom, et le nom de sa filière.
2. De récupérer les moyennes générales de chaque élèves, même ceux qui n'ont aucune note d'enregistrée. (utilisation d'un GROUP BY essentielle !). Les colonnes qui ressortiront de votre requête devront être : MatriculeEleve, Nom, Prénom, Moyenne (la moyenne est un alias)
3. De récupérer les enseignants qui dirigent une filière avec le nom de la filière (Nom, Prénom de l'enseignant, et Nom de la filière)
4. De récupérer pour chaque filières, le nombre de cours suivis par celle-ci (Nom de la filière, Nombre de cours)
5. De récupérer pour chaque filières, une liste des cours suivis par celle-ci (Nom de la filière, Intitulé du cours)
6. De récupérer pour chaque enseignant, le nombre de cours qu'il enseigne (Nom de l'enseignant, Nombre de cours qu'il enseigne)
7. De récupérer pour chaque enseignant, l'intitulé du cours qu'il enseigne (Nom de l'enseignant, Intitulé du cours)
8. De récupérer pour chaque cours, le nom de sa filière et le nom et prénom de l'enseignant qui l'enseigne
9. De récupérer pour chaque enseignant, l'intitulé du cours et le nom de la filière de ce cours
10. De récupérer pour chaque élève son nom, son prénom, le nombre de notes qu'il a, l'intitulé de sa filière, et le nombre de cours reliés à sa filière

# Exercice sur les jointures - suite

Dans la base entrepôt, rédiger les requêtes qui permettent :

1. De lister chaque nom de produit avec chaque nom de catégories pour ce produit
2. De lister pour chaque catégories, le nombre de produits associés

Dans la base animalerie, rédiger les requêtes qui permettent :

3. De lister chaque animal avec le nom de sa race et de son espèce
4. De lister pour chaque achat le nom de l'acheteur, la race de l'animal et son espèce
5. De lister pour chaque espèces le nombre d'animaux existants dans l'animalerie

Dans la base films, rédiger les requêtes qui permettent :

6. De lister pour chaque rôle le nom de l'acteur et le nom du film (les acteurs sont référencés dans la table Artistes)
7. De lister pour chaque pays le nombre de films reliés à ce pays
8. De lister pour chaque réalisateurs le nombre de films qu'ils ont réalisés (les réalisateurs sont référencés dans la table Artistes)

Dans la base bibliotheque, rédiger les requêtes qui permettent :

9. De lister pour chaque livre le nom du livre, le nom du genre, et le nom et prénom de l'auteur
10. De lister le nombre de livres empruntés pour chaque clients