

SQL – Structured Query Language



- > **INFRASTRUCTURES** SYSTÈMES & RÉSEAUX
- > **CYBERSÉCURITÉ** INFRASTRUCTURES & APPLICATIONS
- > **DEVOPS / SCRIPTING** & AUTOMATISATION
- > **DEVELOPPEMENT** WEB & MOBILE
- > **TRANSFORMATION NUMERIQUE DES ENTREPRISES**

www.adrar-numerique.com

SQL – Structured Query Language

Pré-requis – Outils et logiciels

Pour la suite de cette formation, vous allez devoir installer quelques outils

1. Serveur de Base de Données

- La solution « simple » sur **Windows** : Installer **WampServer** ou **XAMPP**
Cela vous fournira un serveur LAMP fonctionnel (Apache, MySQL, PHP),
- La solution d'avenir : utiliser **Docker**, un gestionnaire de conteneurs.
C'est la solution que vous aurez le plus de chances de croiser en entreprise.

La courbe d'apprentissage est un peu plus ardue que pour la solution précédente, mais lorsque vous avez une config fonctionnelle, c'est beaucoup plus efficace et souple.

Docker est une solution **multi-plateformes**, fonctionnant aussi bien sous Windows, Linux ou Mac.

Installation de Docker sous Windows

1. Installer Docker pour Windows (<https://www.docker.com/>)
2. Installer WSL 2 (Windows Subsystem for Linux)
 - Ouvrir un terminal powershell et entrer les commande suivante :
wsl.exe --update
Puis :
wsl --set-default-version 2
 - Ajouter votre utilisateur local au groupe docker-users
 - Ouvrez un terminal PowerShell en tant qu'administrateur et entre la commande suivante :
net localgroup docker-users "your-user-id" /ADD
 - Fermez votre session et rouvrez là

SQL – Structured Query Language

Installation de Docker sous Windows

- Créez un répertoire qui sera votre répertoire de travail, puis clonez-y le dépôt git suivant :
git clone <https://github.com/guirod/docker-lamp.git>
Ce dépôt contiendra l'essentiel pour la suite de votre formation (serveur MySQL, serveur Apache, PHP 8.1, serveur SMTP + mailcatcher) et sera mis à jour au besoin
- Quelques commandes utiles :
 - **Lancer et couper vos containers (serveurs) :**
docker-compose up -d
docker-compose down
 - **Lister les containers**
docker container ls
 - **Obtenir une connexion en root sur le container**
docker exec -it nom_du_container executable
 - **Exemple pour se connecter au container apache et lancer l'exécutable "bash"**
docker exec -it docker-lamp-docker-lamp-php-1 bash

SQL – Structured Query Language

Installation d'une GUI (Graphic User Interface) pour ses bases de données

- 2 principaux outils, je vous conseille d'installer les 2.
 - MySQL Workbench
 - <https://www.mysql.com/fr/products/workbench/>
 - Graphiquement plus agréable
 - Parfois un peu lourd et buggué pour certaines tâches (import et export notamment)
 - HeidiSQL
 - <https://www.heidisql.com/>
 - Un peu moins convivial, mais très puissant.
 - Compatible avec de nombreux SGBDr (MySQL/MariaDB, PostgreSQL, SQL Server)
- Pour chacun de ces outils, configurez et sauvegarder votre connexion :
 - Host : localhost / 127.0.0.1
 - Username : root
 - Password : p@ssw0rd pour les utilisateurs Docker (configuré dans le fichier compose.yaml)

SQL – Structured Query Language

Connexion en ligne de commande

- Se connecter au container MySQL

```
docker exec -it docker-lamp-docker-lamp-mysql-1 bash
```

- Se connecter avec son utilisateur

```
mysql -hlocalhost -uroot -pp@ssw0rd (problème, le pass sera dans votre bash history)  
mysql -h localhost -u root -p (un prompt vous demandera votre mdp)
```

- Enjoy (Non. Mais parfois vous n'aurez pas le choix)

SQL – Structured Query Language

SQL Introduction

SQL – Structured Query Language

SQL – Structured Query Language

- Langage informatique normalisé
- N'est pas un langage procédural
- Créé par IBM en 1974
- Normalisé depuis 1986
 - Recommandation de l'ANSI
 - Norme ISO/CEI 9075 - Technologies de l'information - Langages de base de données SQL

SQL – Structured Query Language

SQL – Structured Query Language

Le langage SQL permet principalement de faire du CRUD :

- Create : Ecrire, insérer des données
- Read : Lire des données
- Update : Modifier des données
- Delete : Supprimer des données

Il permet aussi de modifier la structure de la base de données :

- Créer, modifier, supprimer des bases de données
- Créer, modifier, supprimer des tables
- Créer, modifier, supprimer des utilisateurs et gérer leurs privilèges

SQL – Structured Query Language

Base De Données Relationnelle

- 3 objectifs :
 - **Garantir l'intégrité des données**
 - Eviter l'altération des données (usure, pannes, erreurs, malveillance)
 - Eviter l'incohérence des données
 - La duplication des données : exemple 2 adresses différentes pour un utilisateur
 - Les valeurs aberrantes : exemple Age < 0
 - **Garantir l'indépendance entre données et traitements**
 - L'information correspondant à une donnée doit être compréhensible indépendamment
 - Si une donnée est calculée, on évitera de la stocker en BDD
 - **Traitements rationalisés**
 - Une fois les données définies, les traitements consistent principalement à ajouter, modifier, supprimer et consulter des données

SQL – Structured Query Language

SGBDr – Système de Gestion de Base de Données Relationnelles

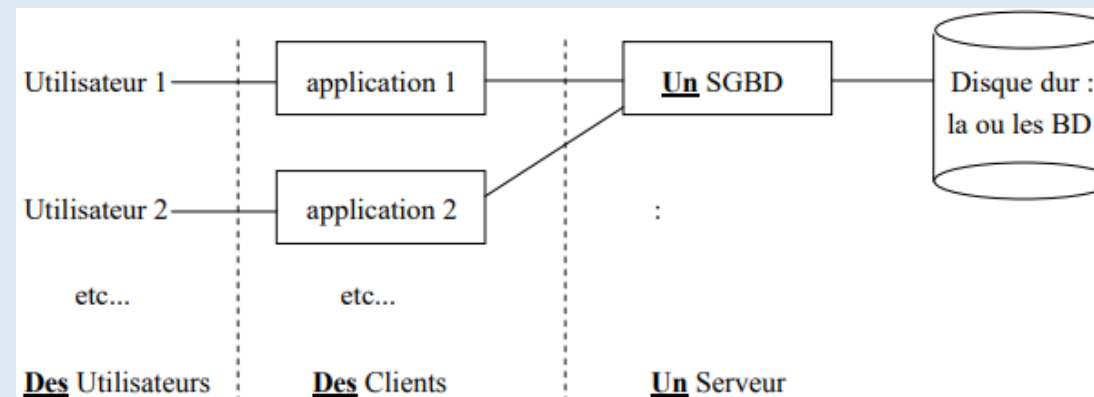
- Ensemble de logiciels faisant l'interface entre l'utilisateur et la BDD
- **Objectifs :**
 - Stockage et requêtage des données
 - Multi-utilisateurs => Gestion des droits d'accès, des collisions
 - Limiter la redondance (duplication de données) => Intégrité des données

SQL – Structured Query Language

SGBDr – Système de Gestion de Base de Données Relationnelles

■ Architecture

- Le plus souvent on utilise une architecture client/serveur



- Une BDD est stockée sur 1 ou plusieurs disques durs
- 1 seul SGBD par BDD
- Plusieurs applications / API (clients) peuvent communiquer avec le SGBD

SQL – Structured Query Language

SGBDr – Système de Gestion de Base de Données Relationnelles

- L'objectif majeur du SGBD étant d'assurer l'intégrité, voici quelques moyens mis en œuvre
 - Conformité au modèle : Vérifier que les données saisies soient cohérentes
 - Accès concurrents : Eviter les incohérences dues à des modifications multiples au même moment (transactions)
 - Gestion sur panne : Garantir le maintien de la cohérence même quand une panne intervient
 - Autorisations d'accès

SQL – Structured Query Language

SGBDr – Système de Gestion de Base de Données Relationnelles

- Les SGBDr les plus connus :
 - MySQL (Oracle) / MariaDB (Fork open source de MySQL)
 - OracleDB
 - PostgreSQL : alternative open source à OracleDB
 - SQL Server (Microsoft)

SQL – Structured Query Language

SQL

Administration d'une BDD

SQL – Structured Query Language

Création de BDD

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name [create_option]...  
create_option: [DEFAULT] {  
    CHARACTER SET [=] charset_name |  
    COLLATE [=] collation_name |  
    ENCRYPTION [=] {'Y' | 'N'}  
}
```

IF NOT EXISTS : Si cette option est présente, cette commande ne lèvera pas d'exception si la BDD existe déjà. Il n'y fera aucune modification.

create_option permet de définir des paramètres de la BDD.

Pour plus d'informations : <https://dev.mysql.com/doc/refman/8.0/en/create-database.html>

SQL – Structured Query Language

Suppression et modification de BDD

- SUPPRESSION

DROP {DATABASE | SCHEMA} [IF EXISTS] *db_name*;

- MODIFICATION

ALTER {DATABASE | SCHEMA} [*db_name*] *alter_option* ...

alter_option: { [DEFAULT] CHARACTER SET [=] *charset_name* |
[DEFAULT] COLLATE [=] *collation_name* |
[DEFAULT] ENCRYPTION [=] {'Y' | 'N'} |
READ ONLY [=] {DEFAULT | 0 | 1} }

- SELECTIONNER UNE BDD : Nécessaire pour pouvoir effectuer des opérations (création ou consultation)

USE *db_name*;

- DIVERS

SHOW DATABASES;

SQL – Structured Query Language

Gestion des droits et utilisateurs

■ Sécurité

- Limiter au strict nécessaire les droits des utilisateurs exposés (application web)
- Utiliser des mots de passe forts et uniques
- Maintenir ses logiciels à jour

SQL – Structured Query Language

Gestion des droits et utilisateurs

- Création d'un utilisateur
`CREATE USER 'alice'@'localhost' IDENTIFIED BY 'P@sswOrd';`
- Afficher les utilisateurs
`SELECT * FROM mysql.user;`
- Assigner les privileges
`GRANT PRIVILEGE ON database.table TO 'username'@'host';`

Exemple :

`GRANT CREATE, ALTER, DROP, INSERT, UPDATE, DELETE, SELECT, REFERENCES, RELOAD
ON *.* TO 'alice'@'localhost' WITH GRANT OPTION;`

- Révoquer un privilege
`REVOKE type_of_permission ON database_name.table_name FROM 'username'@'host'`

SQL – Structured Query Language

Gestion des droits et utilisateurs

- Assigner tous les privileges à un utilisateur
GRANT ALL PRIVILEGES ON database.table TO 'username'@'host';
- Appliquer les modifications:
FLUSH PRIVILEGES;
- Afficher les privileges d'un utilisateur
SHOW GRANTS FOR 'username'@'host';
- Supprimer un utilisateur
DROP USER 'username'@'localhost';

SQL – Structured Query Language

Exercice

- Coder un script pour :
Créer un autre utilisateur 'bob'
Créer une BDD pour bob lui assignant tous les privileges
- Ressource :
<https://www.digitalocean.com/community/tutorials/how-to-create-a-new-user-and-grant-permissions-in-mysql>

SQL – Structured Query Language

SQL

Structure des données

SQL – Structured Query Language

Les Tables

- Les tables sont les objets qui contiennent toutes les données d'une BDD.
- Données organisées dans un tableau à deux dimensions
 - Les lignes sont des tuples, enregistrements ou n-uplets
 - Les colonnes sont des attributs

SQL – Structured Query Language

Les Tables

| id | nom | prénom | profession | code postal | ville |
|----|---------|----------|----------------------|-------------|------------|
| 1 | Durand | Michel | Directeur | 75016 | Paris |
| 2 | Dupond | Karine | Secrétaire | 92000 | Courbevoie |
| 3 | Mensoif | Gérard | Commercial | 75001 | Paris |
| 4 | Monauto | Alphonse | Commercial | 75002 | Paris |
| 5 | Emarre | Jean | Employé | 75015 | Paris |
| 6 | Abois | Nicole | Secrétaire | 95000 | St Denis |
| 7 | Dupond | Antoine | Assistant commercial | 75014 | Paris |

SQL – Structured Query Language

Les types de données

- Numériques exacts :
 - Entiers signés :
 - INTEGER 4 octets => -2147483648 à +2147483647
 - SMALLINT (2 octets), BIGINT (8 octets)
 - Décimaux signés :
 - NUMERIC, DECIMAL
- Numériques approximatifs
 - REAL, DOUBLE PRECISION, FLOAT
- Chaînes de caractères
 - Longueur fixe : CHAR
 - Longueur variable : VARCHAR
 - Objets de type caractère : CLOB

SQL – Structured Query Language

Les types de données

- Chaînes binaires
 - Nombre d'octets fixe : BINARY
 - Longueur variable : VARBINARY
 - Binary Large Object : BLOB
- Booléens
 - 3 valeurs : true, false, unknown
- Dates et heures
 - DATE
 - TIME/TIMESTAMP WITH/WITHOUT TIMEZONE
- DIVERS
 - XML, ARRAY, INTERVAL

SQL – Structured Query Language

Opérations sur les tables

- Création

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
    (create_definition,...)  
    [table_options]  
    [partition_options]
```

Exemple :

```
CREATE TABLE users (  
    id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    name VARCHAR(100),  
    email VARCHAR(255) NOT NULL,  
    birthdate DATE  
);
```

<https://dev.mysql.com/doc/refman/8.0/en/create-table.html>

SQL – Structured Query Language

Opérations sur les tables

- Modification :
 - Ajouter/Modifier/Supprimer un attribut
 - Ajouter/Modifier/Supprimer une contrainte
 - Modifier des propriétés de la table

ALTER TABLE *tbl_name*
 [*alter_option* [, *alter_option*] ...]
 [*partition_options*]

Exemple :

ALTER TABLE *users*
 ADD *firstname* VARCHAR(100) **AFTER** *name*;

<https://dev.mysql.com/doc/refman/8.0/en/alter-table.html>

SQL – Structured Query Language

Indexes

- Dans chaque table, on peut définir un ou plusieurs indexes.
- Un ROWID est créé pour chaque enregistrement
- Avantage : Requêtes plus rapides
- Inconvénient : Peuvent surcharger le système donc à utiliser avec précaution
- Exemple :

Création d'un index : Notez que l'on peut le définir sur une ou plusieurs colonnes

CREATE INDEX index_name **ON** table_name (column1, column2, ...);

Suppression d'un index

ALTER TABLE table_name **DROP INDEX** index_name;

SQL – Structured Query Language

Contraintes

Les contraintes sont des règles appliquées aux attributs d'une table. Elles permettent d'améliorer l'intégrité des données de la base.

- Les contraintes les plus communes sont :
 - **NOT NULL** : Contraint à avoir une valeur différente de NULL.
 - **DEFAULT** : Fournit une valeur par défaut à la colonne.
 - **UNIQUE** : Ne permet pas que 2 enregistrements aient la même valeur pour une colonne (email)
 - **CHECK** : Active une condition pour qu'un enregistrement soit valide.
 - **INDEX**
 - **PRIMARY KEY** : La clé primaire permet d'identifier de manière unique chaque enregistrement. Une clé primaire peut être composée, elle fait dans ce cas référence à plusieurs attributs de la table.
 - **FOREIGN KEY** : Une clé étrangère permet de relier 2 tables.

SQL – Structured Query Language

Contraintes - Clés

- **Clé primaire :**

Une clé primaire permet d'identifier chaque enregistrement d'une table.

Par définition, elle doit être UNIQUE et NOT NULL.

Une clé primaire peut être simple (définie par un seul attribut) ou composée de plusieurs attributs.

- **Clé étrangère :**

Une clé étrangère est une contrainte qui permet d'associer plusieurs tables entre elles tout en assurant l'intégrité des données.

Une clé étrangère fait référence à une clé primaire d'une autre table.

A noter :

- Il est impossible de faire référence à une clé primaire si celle-ci n'existe pas dans la BDD.
- Il n'est par défaut pas possible de supprimer un enregistrement si sa clé primaire est référencée comme clé étrangère dans une autre table.

SQL – Structured Query Language

Contraintes – ON DELETE / ON UPDATE

- Clé étrangère :

Si un enregistrement est référencé dans une autre table, le SGBD empêchera la modification ou suppression de la clé primaire.

Si l'on souhaite forcer la modification / suppression, il est nécessaire de le renseigner lors de la création de la clé secondaire.

Pour cela on doit utiliser les instructions ON DELETE et ON UPDATE

- ON DELETE : A la suppression, effectue une action sur tous les enregistrements enfants.
- ON UPDATE : A la modification, reporte le changement sur tous les enregistrements enfants.

L'usage le plus courant est d'utiliser ON DELETE CASCADE ou ON UPDATE CASCADE.

Toutefois, c'est un comportement assez destructeur ce qui n'est pas toujours souhaitable.

Dans ce cas, on peut utiliser des alternatives, telles que :

ON DELETE | UPDATE [SET NULL | SET DEFAULT | NO ACTION | CASCADE]

SQL – Structured Query Language

Contraintes - Exemple

```
CREATE TABLE employes(  
    id INT NOT NULL,  
    nom VARCHAR (100) NOT NULL,  
    email VARCHAR (100) NOT NULL UNIQUE,  
    age INT NOT NULL CHECK (age >= 18),  
    salaire DECIMAL (18, 2) DEFAULT 3000.00,  
    PRIMARY KEY (id)  
);  
CREATE TABLE congés (  
    id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    date_debut DATE NOT NULL,  
    date_fin DATE NOT NULL,  
    id_employe INT,  
    FOREIGN KEY (id_employe) REFERENCES employes(id) ON DELETE CASCADE  
    -- Pour pouvoir nommer la contrainte, utiliser la syntaxe suivante  
    CONSTRAINT fk_id_employe FOREIGN KEY (id_employe) REFERENCES employes(id)  
);
```

SQL – Structured Query Language

Contraintes - Suppression

- En ciblant la contrainte :

ALTER TABLE table_name

ALTER COLUMN column_name **DROP CONSTRAINT**;

Ex : **ALTER TABLE** employe

ALTER COLUMN salaire **DROP DEFAULT**;

- Si la contrainte a un alias

ALTER TABLE table_name **DROP CONSTRAINT** constraint_alias;

SQL – Structured Query Language

Associations

Lorsque l'on modélise notre BDD, on va identifier différents types d'associations qui vont lier les tables entre elles.

Bien concevoir ses associations est indispensable pour satisfaire les conditions des formes normales et assurer l'intégrité des données.

On dénombre **5 types de relations** :

- One-to-One
- One-to-Many / Many-to-One
- Many-to-Many
- Self-Reference (association récursive)

SQL – Structured Query Language

Associations – One-to-One

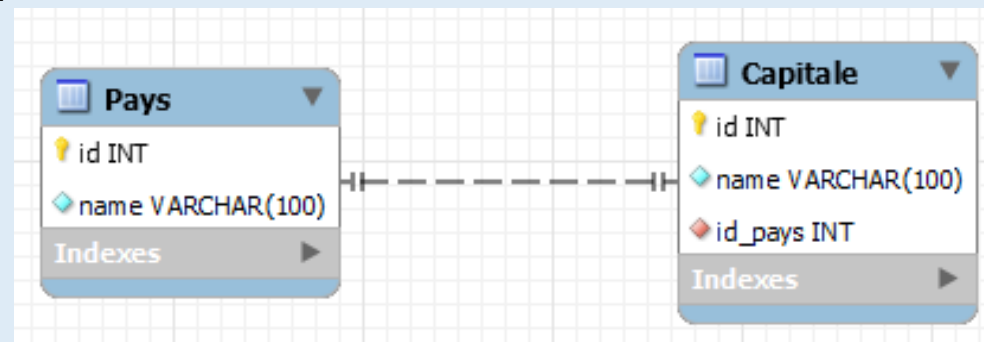
Permet d'associer un et un seul enregistrement d'une table T1 à un et un seul enregistrement d'une table T2.

Implémentée par l'utilisation d'une clé étrangère.

- Veiller à ajouter des contrôles (code et/ou BDD) pour assurer l'intégrité des données (ex : contrainte d'unicité sur la clé étrangère)
- Cette contrainte sera rarement utilisée, car généralement, la donnée peut directement être ajoutée dans la table « parent »

Exemples :

- Citoyen <-> Passeport
- Capitale <-> Pays



SQL – Structured Query Language

Associations – One-to-Many et Many-to-One

One-to-Many :

Associe un enregistrement d'une table T1 à de multiples enregistrements d'une table T2

Many-to-One :

Associe de multiples enregistrements d'une table T1 à un unique enregistrement d'une table T2

Implémentée par l'utilisation d'une clé étrangère.

- Veiller à ajouter des contrôles (code et/ou BDD) pour assurer l'intégrité des données.

Exemple :

- Pays <-> Ville : Un pays possède plusieurs villes, une ville n'appartient qu'à un seul pays



SQL – Structured Query Language

Associations – Many-to-Many

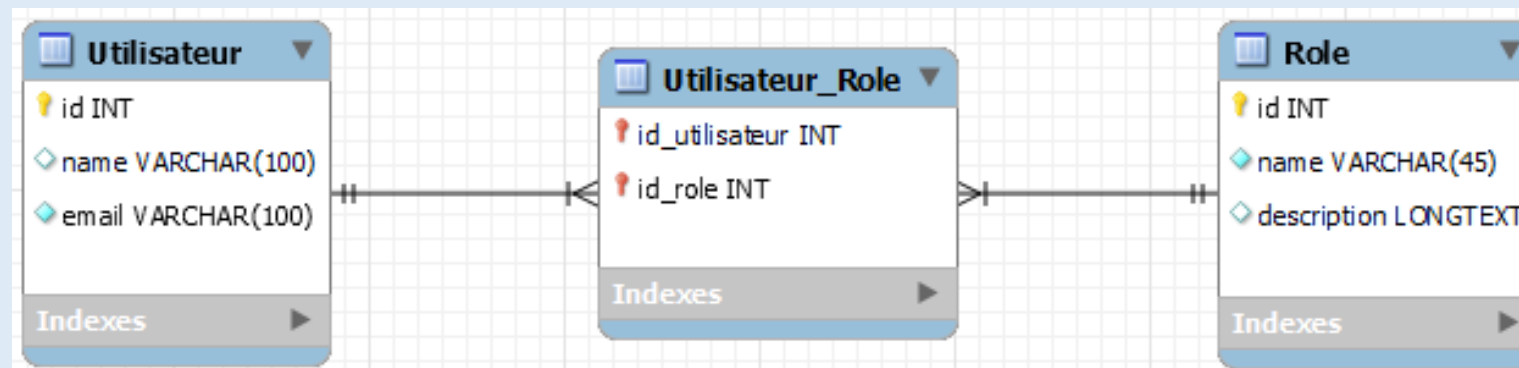
Associe un enregistrement d'une table T1 à de multiples enregistrements de la table T2 et vice versa.

Pour implémenter une association Many-to-Many, il est nécessaire de passer par une **table d'association**.

La table d'association peut aussi être **porteuse de donnée**, par exemple ci-dessous, la table Utilisateur_Role pourrait indiquer la date d'expiration du rôle.

Exemple :

- Utilisateur <-> Rôle : un utilisateur peut avoir plusieurs rôles et un rôle peut être possédé par plusieurs utilisateurs.



SQL – Structured Query Language

Associations – Self-Reference

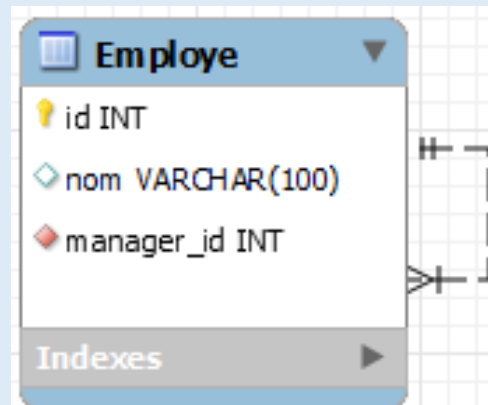
Une association Self-Reference permet d'associer un enregistrement d'une table T à un autre enregistrement d'une table T.

L'association sera souvent une association One-to-Many ou Many-to-One.

C'est notamment utilisé lorsque l'on souhaite gérer une arborescence.

Exemples :

- Catégories <-> Catégorie enfant
- Employé <-> Manager



SQL – Structured Query Language

SQL Ressources

SQL – Structured Query Language

Documentation

- Site officiel :
<https://www.mysql.com/>
- Documentation officielle :
<https://dev.mysql.com/doc/refman/8.0/en/>
- Tuto W3Schools :
<https://www.w3schools.com/sql/default.asp>
- MySQL Tutorial :
<https://www.mysqltutorial.org/>
- Slide sur l'optimisation :
<https://www.slideshare.net/ZendCon/joinfu-the-art-of-sql-tuning-for-mysql-presentation>

SQL – Structured Query Language

SQL Travaux Pratiques

SQL – Structured Query Language

SQL

Insertion et modification de données

SQL – Structured Query Language

Insertion d'enregistrements

Maintenant que nous avons une structure de BDD cohérente, nous pouvons commencer à insérer des données dans notre base.

Pour insérer des données, nous allons utiliser la commande « INSERT INTO ».

Nous allons détailler sa syntaxe.

SQL – Structured Query Language

INSERT INTO

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name
      [PARTITION (partition_name [, partition_name] ...)]
      [(col_name [, col_name] ...)]
      { {VALUES | VALUE} (value_list) [, (value_list)] ... }
      [AS row_alias[(col_alias [, col_alias] ...)]]
      [ON DUPLICATE KEY UPDATE assignment_list]
```

Voici le lien vers la documentation officielle : <https://dev.mysql.com/doc/refman/8.0/en/insert.html>

Vous pourrez avoir des détails sur les différentes options.

SQL – Structured Query Language

INSERT INTO

Comme précédemment, la syntaxe semble barbare, mais la syntaxe que nous utiliserons le plus souvent sera beaucoup plus digeste :

```
INSERT INTO tbl_name(col_1, col_2, ...)  
VALUES (value_col_1, value_col_2, ...);
```

- **tbl_name** correspond au nom de notre table.
- **Col_1, col_2, ...** : les noms de colonne pour lesquelles nous souhaitons préciser les valeurs de notre enregistrement
- **value_col_1, value_col_2, ...** : *les valeurs que nous voulons insérer.*

Notez qu'il est possible de ne pas préciser la liste des colonnes, mais dans ce cas il faudra envoyer les valeurs pour toutes les colonnes de la table, dans l'ordre. Je ne suis personnellement pas fan de cette syntaxe.

SQL – Structured Query Language

INSERT INTO

Il est également possible de faire de l'insertion « de masse », en envoyant plusieurs listes de valeurs, séparées d'une virgule. On procéderait de la façon suivante :

```
INSERT INTO tbl_name(col_1, col_2)  
VALUES (value_col_1_e1, value_col_2_e1), (value_col_1_e1, value_col_2_e1),  
(value_col_1_e1, value_col_2_e1);
```

SQL – Structured Query Language

INSERT INTO : Exemple

- Insertion d'enregistrement unique :

```
INSERT INTO users(`name`,firstname,email)  
VALUES('Cartman', 'Eric', 'eric@cartman.com');
```

- Insertion multiple d'enregistrements :

```
INSERT INTO users (`name`,firstname,email)  
VALUES  
('McCormick', 'Kenny', 'kenny@mccormick.com'),  
('Broflovski', 'Kyle', 'kyle@broflovski.com'),  
('Marsh', 'Stan', 'stan@marsh.com'),  
('Garrison', 'Herbert', 'herbert@garrison.com');
```

UPDATE : Modification d'enregistrements

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference  
    SET assignment_list  
    [WHERE where_condition]  
    [ORDER BY ...]  
    [LIMIT row_count]
```

Documentation : <https://dev.mysql.com/doc/refman/8.0/en/update.html>

SQL – Structured Query Language

UPDATE

Ici encore, le plus souvent nous utiliserons une syntaxe allégée :

```
UPDATE table_reference  
  SET col1 = val_col1, col2 = val_col2, ...  
  [WHERE where_condition]
```

La clause “where” va nous permettre de cibler des enregistrements particuliers selon des critères.

Lorsque la clause “where” est absente d’une requête d’update, toutes les données de la table seront modifiées en conséquence.

Il faut donc être très prudent lors de l’exécution de ce genre de requête.

SQL – Structured Query Language

UPDATE : Exemple d'utilisation

Ajout d'une valeur à la colonne « birthdate » de tous les users.

```
UPDATE users  
SET birthdate = '1997-08-13';
```

Modification de l'adresse mail d'un utilisateur en le recherchant par son nom et prénom.

```
UPDATE users  
SET email = 'eric.cartman@south-park.com'  
WHERE `name` = 'Cartman' AND firstname = 'Eric';
```

Pour visualiser les résultats :

```
SELECT * FROM users;
```

SQL – Structured Query Language

DELETE : Suppression d'enregistrements

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name [[AS] tbl_alias]  
  [PARTITION (partition_name [, partition_name] ...)]  
  [WHERE where_condition]  
  [ORDER BY ...]  
  [LIMIT row_count]
```

Documentation : <https://dev.mysql.com/doc/refman/8.0/en/delete.html>

SQL – Structured Query Language

DELETE : Suppression d'enregistrements

Ici encore, la commande allégée ressemblera à :

DELETE FROM *tbl_name* **WHERE** *where_condition*

Là encore, la clause WHERE est facultative. Si elle n'est pas renseignée (et que votre serveur MySQL le permet), toutes les données de la table seraient supprimées.

Il faut donc être extrêmement prudent lors de l'utilisation de cette commande.

SQL – Structured Query Language

DELETE : Exemples

- Suppression d'un enregistrement, ciblé par la primary key :

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

- *Suppression d'enregistrements dans une plage d'id :*

```
DELETE FROM users WHERE id BETWEEN 1 AND 3;
```

SQL – Structured Query Language

DELETE : Exemples

- Nous avons vu que la commande DELETE pouvait purger totalement une table.
- Cela peut parfois être utile. Toutefois, cela ne va pas reset le compteur “auto_increment” utilisé dans les identifiants.
- Si l’on souhaite reset le compteur d’auto_increment, il conviendra d’utiliser la commande “TRUNCATE” :

TRUNCATE TABLE users;

SQL – Structured Query Language

SQL

Opérateurs logiques

SQL – Structured Query Language

Opérateurs logiques

Nous avons commencé à manipuler quelques opérateurs logiques dans les clauses WHERE.

MySQL permet l'utilisation de nombreux opérateurs logiques. Nous allons en étudier quelques uns

SQL – Structured Query Language

Opérateurs

| Opérateur | Description | Datatype concerné |
|----------------------|----------------------------------|-------------------|
| = | Opérateur d'égalité | Tous |
| <>, != | Différent | Tous |
| > | Supérieur | Numérique |
| < | Inférieur | Numérique |
| >= | Supérieur ou égal | Numérique |
| <= | Inférieur ou égal | Numérique |
| BETWEEN x and y | Compris entre x et y | Numérique, dates |
| IN(liste de valeurs) | Appartient à la liste de valeurs | Tous |
| IS NULL | Valeur est null | Tous |

SQL – Structured Query Language

Opérateurs

| Opérateur | Description | Datatype concerné |
|-----------|----------------------------------------------------------------------------------------------------------------------------|-------------------|
| NOT | Opérateur logique de négation Renvoie true il évalue false, et renvoie false si il évalue true Exemple : IS NOT NULL | Opérateur logique |
| AND | Opérateur ET logique Ex : A AND B = true si A = true ET B = true | Opérateur logique |
| OR | Opérateur OU logique EX : A OR B = true si A = true ET/OU B = true | Opérateur logique |
| XOR | Opérateur OU exclusif EX : A XOR B = true si A = true OU B = true mais pas si les deux sont true | Opérateur logique |

SQL – Structured Query Language

Opérateurs

| Opérateur | Description | Datatype concerné |
|-----------|----------------------------------------------------------------------------------------------------------------------------|-------------------|
| NOT | Opérateur logique de négation Renvoie true il évalue false, et renvoie false si il évalue true Exemple : IS NOT NULL | Opérateur logique |
| AND | Opérateur ET logique Ex : A AND B = true si A = true ET B = true | Opérateur logique |
| OR | Opérateur OU logique EX : A OR B = true si A = true ET/OU B = true | Opérateur logique |
| XOR | Opérateur OU exclusif EX : A XOR B = true si A = true OU B = true mais pas si les deux sont true | Opérateur logique |

SQL – Structured Query Language

Opérateur « like »

- L'opérateur like est utilisé pour rechercher une chaîne de caractères dans une autre.
- Cet opérateur est très utilisé pour filtrer un tableau de résultats.
- Il ne prend pas en compte la casse. Ainsi : « Cheval » LIKE « CHeVaL » vaudra true
- On l'utilise en combinaison avec le symbole '%', qui représente 0, 1 ou n caractères

| Opérateur | Description | Datatype concerné |
|------------|--------------------------------------------------------------------------------|-----------------------|
| X LIKE Y | La chaîne de caractères X devra être identique à Y, à l'exception de la casse. | Chaines de caractères |
| X LIKE %Y | La chaîne Y devra se terminer par la chaîne X | Chaines de caractères |
| X LIKE Y% | La chaîne Y devra commencer par la chaîne X | Chaines de caractères |
| X LIKE %Y% | La chaîne X devra être comprise dans la chaîne Y | Chaines de caractères |

SQL – Structured Query Language

Opérateur « like » : exemples

Une démo vaut plus que de long discours.

Je vous invite donc à aller regarder et manipuler l'opérateur LIKE pour connaître un peu mieux son fonctionnement, il vous sera très utile.

Documentation et exemples :

- https://www.w3schools.com/mysql/mysql_like.asp
- <https://sql.sh/cours/where/like>

SQL – Structured Query Language

Précédence des opérateurs

- Comme en mathématiques, les opérateurs n'ont pas tous la même priorité lors de l'évaluation d'une opération logique.
- Voici la liste des opérateurs, de la priorité la plus basse à la plus haute.

:=
||, OR, XOR
&&, AND
BETWEEN, CASE, WHEN, THEN, ELSE
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
|
&
<<, >>
-, +
*, /, DIV, %, MOD
^
- (unary minus), ~ (unary bit inversion)
- NOT, !
- BINARY, COLLATE

SQL – Structured Query Language

SQL

Requêtes de sélection

SQL – Structured Query Language

SELECT

La commande SELECT est la commande qui permet de nous retourner les enregistrements de nos bases de données.

Cette commande est complète et complexe. Comme précédemment, voici l'extrait de la documentation MySQL.

Cette fois-ci, je vous épargne la documentation officielle, assez indigeste.

Je vous invite toutefois à aller y jeter un œil :

<https://dev.mysql.com/doc/refman/8.0/en/select.html>

SQL – Structured Query Language

SELECT

- **Commande SELECT « basique »**

```
SELECT nom_colonne FROM nom_table;
```

Cette commande retournera pour toutes les lignes (enregistrements) de la table, la valeur de la colonne « nom_colonne ».

- **Récupérer la valeur de plusieurs colonnes** : pour cela il suffit de lister toutes les colonnes que nous souhaitons récupérer. Exemple :

```
SELECT `name`, firstname FROM users;
```

- **Récupérer la valeur de toutes les colonnes** pour tous les enregistrements de la table :

```
SELECT * FROM users;
```

SQL – Structured Query Language

SELECT

- **Commande SELECT « basique »**

```
SELECT nom_colonne FROM nom_table;
```

Cette commande retournera pour toutes les lignes (enregistrements) de la table, la valeur de la colonne « nom_colonne ».

- **Récupérer la valeur de plusieurs colonnes** : pour cela il suffit de lister toutes les colonnes que nous souhaitons récupérer. Exemple :

```
SELECT `name`, firstname FROM users;
```

- **Récupérer la valeur de toutes les colonnes** pour tous les enregistrements de la table :

```
SELECT * FROM users;
```

SQL – Structured Query Language

SELECT

- Retourner des enregistrements en supprimant les doublons : SELECT DISTINCT

```
SELECT DISTINCT `name` FROM users;
```

- Voyons maintenant une partie des différentes commandes que peut comprendre un SELECT

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


SQL – Structured Query Language

SELECT

- **Clause WHERE**

C'est la clause que vous utiliserez le plus. Elle permet de filtrer les résultats de la requête en fonction de certains critères.

Par exemple :

```
SELECT * FROM Personne  
WHERE date_naissance BETWEEN '2000-07-02' AND '2014-10-3';
```

Cette requête retournera toutes les personnes dont la date de naissance est comprise entre le 02/07/2000 et le 03/10/2014.

Il est évidemment possible d'utiliser plusieurs critères dans une clause WHERE :

```
SELECT * FROM Personne  
WHERE date_naissance BETWEEN '2000-07-02' AND '2014-10-3'  
AND (email LIKE '%@gmail.com' OR email LIKE '%@hotmail.fr');
```

Cette requête retournera toutes les personnes dont la date de naissance est comprise entre le 02/07/2000 et le 03/10/2014, et qui possèdent une adresse gmail.com ou hotmail.fr (notez l'utilisation des parenthèses)

SQL – Structured Query Language

SELECT

- **Clause ORDER BY**

Cette clause permet de trier les résultats selon certaines colonnes. Pour choisir l'ordre du tri, nous utiliserons les paramètres ASC (croissant) et DESC (décroissant).

Par exemple :

```
SELECT * FROM Personne  
ORDER BY date_naissance ASC;
```

Cette requête retournera tous les enregistrements de la table Personne, triés par la date de naissance (ordre croissant).

On peut aussi trier selon plusieurs colonnes :

```
SELECT * FROM users  
ORDER BY `name` DESC, firstname DESC;
```

Ici, nous trions les résultats, en priorité par nom (ordre décroissant), puis par prénom (décroissant également).

SQL – Structured Query Language

SELECT

- **Clauses LIMIT et OFFSET**

La clause LIMIT permet de limiter le nombre de résultats retournés.

La clause OFFSET permet de retourner les résultats, à partir d'un index déterminé par l'offset.

La conjonction de ces 2 commandes est notamment utilisée pour gérer la pagination d'un tableau de résultats.

```
SELECT * FROM users
LIMIT 5;                // Retourne les 5 1ers résultats uniquement
                        // Nous retournera les enregistrements des id 1 à 5
```

```
SELECT * FROM users
LIMIT 5
OFFSET 5;               // Retourne les 5 1ers résultats, à partir du 5ème
                        // Retournera les résultats des enregistrements 6 à 10
```

SQL – Structured Query Language

SELECT : fonctions d'agrégation

- Les fonctions d'agrégation sont les fonctions qui d'une façon ou une autre, regrouper plusieurs lignes en une seule.
- Les fonctions d'agrégation les plus utilisées sont les suivantes :
 - ❖ AVG(nom_colonne) : calcule la moyenne de plusieurs enregistrements
 - ❖ COUNT(nom_colonne) : retourne le nombre total de lignes retournées
 - ❖ MAX(nom_colonne) : retourne la valeur maximale d'une colonne
 - ❖ MIN(nom_colonne) : retourne la valeur minimale d'une colonne
 - ❖ SUM (nom_colonne) : retourne la somme des enregistrements d'une colonne

SQL – Structured Query Language

SELECT : fonctions d'agrégation

- Exemples avec injection de données aléatoires :

```
CREATE TABLE test (  
id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
val INT  
);
```

```
INSERT INTO test(val)  
VALUES (round(RAND()*100)),(round(RAND()*100)),(round(RAND()*100)),  
(round(RAND()*100)),(round(RAND()*100)),(round(RAND()*100)),  
(round(RAND()*100)),(round(RAND()*100)),(round(RAND()*100)),  
(round(RAND()*100)),(round(RAND()*100)),(round(RAND()*100));
```

```
SELECT COUNT(*) FROM test;  
SELECT SUM(val) FROM test;  
SELECT AVG(val) FROM test;  
SELECT MIN(val) FROM test;  
SELECT MAX(val) FROM test;
```

SQL – Structured Query Language

SELECT : fonctions d'agrégation

- **Clause GROUP BY :**
- Lorsque l'on souhaite récupérer d'autres données de notre table, et utiliser une fonction d'agrégation, il sera nécessaire d'utiliser la clause GROUP BY, qui va regrouper les résultats selon une colonne.
- Imaginons la requête suivante :

```
SELECT nomclient, SUM(product_price) FROM orders;
```

- Si nous exécutons cette commande, nous aurons une erreur SQL disant qu'il manque la clause GROUP BY. En effet, la fonction SUM retournant une seule ligne de résultat, mais pas le « SELECT nomclient » (il y a plusieurs clients), MySQL ne sait pas quel client nous retourner.
- Il va donc falloir lui indiquer clairement que nous souhaitons regrouper les résultats par nomclient. Du coup, la SUM calculée, sera celle concernant chaque client et non la SUM globale de la table.

```
SELECT nomclient, SUM(product_price) AS total_order  
FROM orders  
GROUP BY nomclient;
```

SQL – Structured Query Language

SELECT : fonctions d'agrégation

- Créer un alias : clause AS
- Si on reprend la requête précédente, on peut aussi noter l'utilisation de la clause AS.
- Cette clause permet de définir un alias à la colonne récupérée. Ici, on renomme donc « SUM(product_price) » en « total_order »

```
SELECT nomclient, SUM(product_price) AS total_order  
FROM orders  
GROUP BY nomclient;
```

- Ce nouveau nom de colonne sera celui qui sera retourné à notre logiciel d'administration de BDD (MySQL Workbench, HeidiSQL), mais sera aussi celui retourné à notre programme PHP.

SQL – Structured Query Language

SELECT : fonctions d'agrégation

- Conditions sur les agrégations : clause HAVING
- Retraçons le cheminement d'une requête SQL avec agrégation :
 1. Sélection des colonnes qui seront retournées
 2. Filtrage des lignes retournées par la clause WHERE
 3. Agrégation des résultats selon une fonction
- Une requête SQL ne pouvant posséder 2 clauses WHERE, comment faire un filtre sur le retour d'une fonction d'agrégation ?
- Imaginons que nous souhaitons retourner uniquement les clients dont le montant total de la commande soit > 500. Pour cela, on va utiliser la clause HAVING, qui est l'équivalent d'une clause WHERE qui sera exécutée à la fin de notre requête, après l'agrégation des résultats.

```
SELECT nomclient, SUM(product_price) AS total_order  
FROM orders  
GROUP BY nomclient  
HAVING total_order > 500;
```


SQL – Structured Query Language

SELECT : fonctions d'agrégation

Exemple GROUP BY et HAVING

```
CREATE TABLE test2 (  
  id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
  nomclient VARCHAR(50),  
  val INT  
);
```

```
INSERT INTO test2(nomclient,val)  
VALUES
```

```
('client_1', round(RAND()*100)),('client_1', round(RAND()*100)),('client_1', round(RAND()*100)),  
('client_1', round(RAND()*100)),('client_1', round(RAND()*100)),('client_1', round(RAND()*100)),  
('client_1', round(RAND()*100)),('client_1', round(RAND()*100)),('client_1', round(RAND()*100)),  
('client_2', round(RAND()*100)),('client_2', round(RAND()*100)),('client_2', round(RAND()*100)),  
('client_2', round(RAND()*100)),('client_2', round(RAND()*100)),('client_2', round(RAND()*100)),  
('client_2', round(RAND()*100)),('client_2', round(RAND()*100)),('client_2', round(RAND()*100));
```

SQL – Structured Query Language

SELECT : fonctions d'agrégation

- Exemple GROUP BY et HAVING

```
SELECT nomclient,SUM(val) AS total_order  
FROM test2  
GROUP BY nomclient;
```

```
SELECT nomclient,SUM(val) AS total_order  
FROM test2  
GROUP BY nomclient  
HAVING total_order < 500;           // Adaptez la valeur en fonction des  
données de votre table
```

SQL – Structured Query Language

SQL Jointures

SQL – Structured Query Language

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.

Exemples d'utilisation :

- Imaginons qu'une base de 2 données possède une table "utilisateur" et une 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.
- On peut aussi imaginer qu'un site web possède une table pour les articles (titre, contenu, date de publication ...) et une autre pour les rédacteurs (nom, date d'inscription, date de naissance ...). Avec une jointure il est possible d'effectuer une seule recherche pour afficher un article et le nom du rédacteur. Cela évite d'avoir à afficher le nom du rédacteur dans la table "article".

SQL – Structured Query Language

Types de jointures

Il existe différents types de jointures. Voici les jointures les plus communes :

- **INNER JOIN** : jointure interne pour retourner les enregistrements quand la condition est vrai dans les 2 tables. C'est l'une des jointures les plus communes.
- **LEFT JOIN (ou LEFT OUTER JOIN)** : jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifiée dans l'autre table.
- **RIGHT JOIN (ou RIGHT OUTER JOIN)** : jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifiée dans l'autre table.
- **FULL JOIN (ou FULL OUTER JOIN)** : jointure externe pour retourner les résultats quand la condition est vraie dans au moins une des 2 tables.
- **FULL JOIN (ou FULL OUTER JOIN)** : jointure externe pour retourner les résultats quand la condition est vraie dans au moins une des 2 tables.
- **SELF JOIN** : permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.

SQL – Structured Query Language

Types de jointures

Voici quelques jointures moins utilisées. Nous ne les détaillerons pas ici.

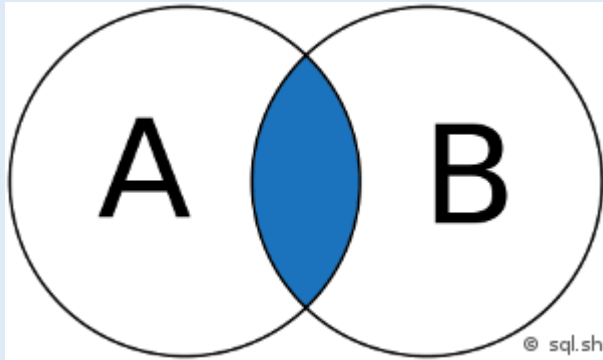
Pour plus d'informations, allez voir ici : <https://sql.sh/cours/jointures>

- **CROSS JOIN** : Produit cartésien de 2 tables : va joindre chaque ligne d'une table avec chaque ligne d'une autre table => renvoie de nombreux résultats.
- **NATURAL JOIN** : jointure naturelle entre 2 tables s'il y a au moins une colonne qui porte le même nom entre les 2 tables SQL
- **UNION JOIN** : jointure d'union

SQL – Structured Query Language

INNER JOIN

Cette jointure retournera l'intersection de 2 tables.



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

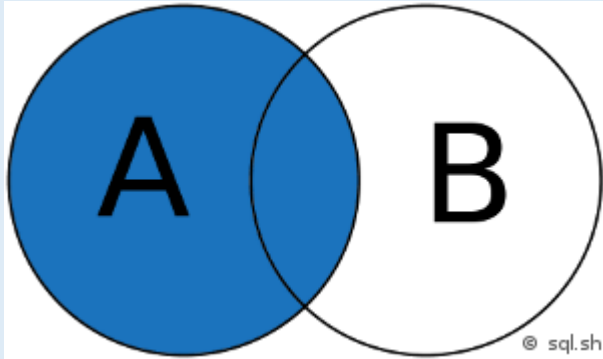
SQL – Structured Query Language

LEFT JOIN

Retourne tous les résultats de A et l'intersection des résultats de A et B

```

SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
    
```

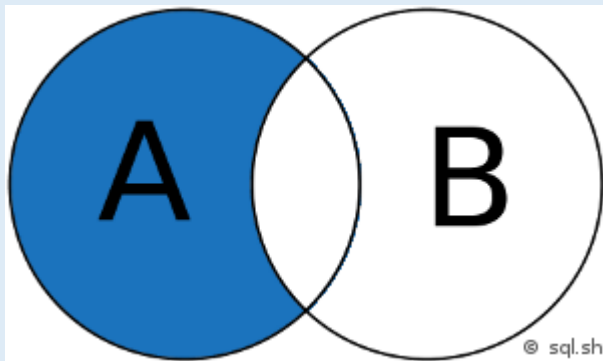


LEFT JOIN sans intersection :

Retourne tous les résultats de A excluant l'intersection de A avec B.

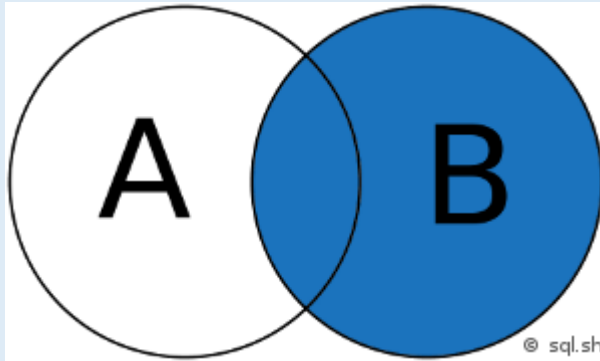
```

SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
WHERE B.key IS NULL
    
```



SQL – Structured Query Language

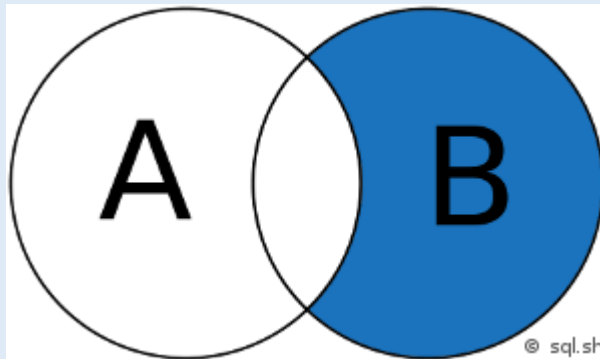
RIGHT JOIN



Retourne tous les résultats de B et l'intersection des résultats de A et B

```

SELECT *
FROM B
LEFT JOIN A ON B.key = A.key
    
```



RIGHT JOIN sans intersection :

Retourne tous les résultats de B excluant l'intersection de A avec B.

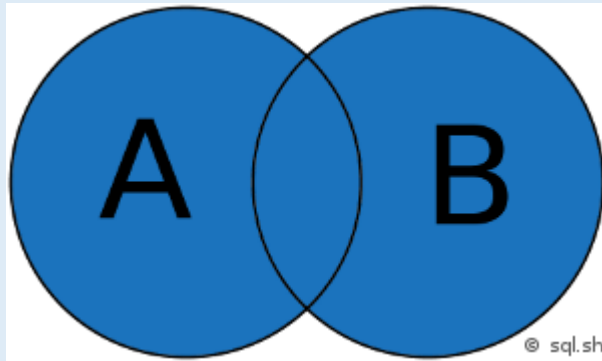
```

SELECT *
FROM B
LEFT JOIN A ON B.key = A.key
WHERE A.key IS NULL
    
```

SQL – Structured Query Language

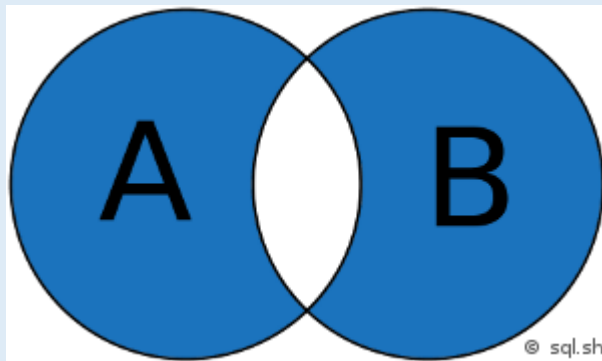
FULL JOIN

Retourne tous les résultats de B et l'intersection des résultats de A et B



```

SELECT *
FROM A
FULL JOIN B ON A.key = B.key
    
```



FULL JOIN sans intersection :

Retourne tous les résultats de A et B excluant l'intersection de A avec B.

```

SELECT *
FROM A
FULL JOIN B ON A.key = B.key
WHERE A.key IS NULL OR B.key IS NULL
    
```

SQL – Structured Query Language

Requêtes imbriquées

Il est possible d’imbriquer des requêtes entre elles.

Cela consiste à utiliser le résultat d’une requête dans un autre.

Voici l’exemple d’un usage assez courant : injection d’un SELECT id d’une table dans un opérateur IN (qui attend une liste).

```
SELECT * FROM articles  
WHERE articles.category_id IN (SELECT id FROM categories);
```