2. Le langage SQL DDL

2.1 Naissance du langage SQL

SQL est né dans les années 1970 dans les **laboratoires de recherche d'IBM**. Il a été conçu sur la base des travaux de E. F. Codd afin de gérer et d'effectuer des recherches dans les bases de données.

Ce n'est pas un langage de programmation comme PHP et Java, mais comme ses initiales l'indiquent **(Structured Query Language)** est un langage de requêtes, dans ce cas, à une base de données.

2.2 Mots réservés

Ils indiquent à la base de données ce qu'il faut faire : créer une table, insérer des données, rechercher une information, etc.

Quelques exemples sont **INSERT, UPDATE, CREATE, TEXT** ou **SELECT**. Peu importe si nous les écrivons en majuscule ou minuscule.

Ces mots sont réservés pour donner des instructions à la base de données afin que vous ne puissiez pas les utiliser comme noms de tables, colonnes, etc.

2.3 Noms

Chaque objet dans la base de données doit avoir un nom et il doit être unique dans sa catégorie. Cela implique qu'il ne peut y avoir deux **tables** portant le même nom, tout comme il ne peut y avoir deux **colonnes** avec le même nom dans une même table.

Comme pour les mots réservés, il importe peu que vous les écriviez en majuscules ou en minuscules.

Règles

- Tout nom doit être une chaine formée de **lettres (sans accent)**, de **chiffres** et de signes **#**, **$** et _.
- Un nom ne peut pas commencer par un chiffre.
- Les mots réservés sont interdits (**insert**, **table**, **text**, **index**, etc).

Il est possible de contourner ces règles. Pour ce faire, il nous suffit d'entourer un nom avec de guillemets ("") ou de crochets ([]). En tout cas, c'est recommandable de s'ajuster aux règles et de ne pas utiliser de guillemets ou de crochets.

2.4 Les colonnes et leur types

Les derniers éléments qui composent le langage sont les types de données, c'est-à-dire les différents types de valeurs que l'on peut enregistrer dans les colonnes d'un tableau.

Voici une liste de quelques types de données sur MySQL:

Type	Exemple
CHAR (20)	'Chaine de texte'
VARCHAR (20)	'Chaine de texte'
SMALLINT, BIGINT ou INTEGER	7500
NUMERIC ou DECIMAL	3425.432
REAL, FLOAT ou DOUBLE PRECISION	6.626E-34
BOOLEAN	TRUE
DATE	'1957-08-14'

Si vous voulez une liste plus exhaustive des types de données, vous pouvez suivre ce lien vers la documentation des types supportés par MySQL.

2.5 Le langage SQL et ses quatre sous-groupes

Les commandes SQL sont divisées en quatre sous-groupes, **DDL**, **DML**, **DCL** et **TCL**.

- **Data Definition Language (DDL)** :
  Un standard pour les commandes qui définissent les différentes structures d'une base de données.
  Les instructions DDL servent à **créer, modifier et supprimer** les objets de base de données tels que les tables, les index et les utilisateurs. Les instructions DDL courantes sont **CREATE, ALTER et DROP.**
- **Data Manipulation Language (DML)** :
  Est la partie du langage SQL qui sert à manipuler des données et inclut la plupart des instructions SQL courantes telles que **SELECT, INSERT, UPDATE, DELETE**, etc. Il est utilisé pour insérer, modifier, récupérer, supprimer et mettre à jour les données dans une base de données.
- **Data Control Language (DCL) :**
  Inclut des commandes telles que **GRANT** et concerne principalement les **droits, permissions** et autres contrôles du système de base de données.
- **Transaction Control Language (TCL) :**
  Inclut des commandes telles que **COMMIT** ou **ROLLBACK** sert à gérer les **transactions** dans une base de données.

Dans ce chapitre nous allons nous concentrer sur le Data Definition Language (DDL) .

2.6 Créer une table

L'instruction **CREATE TABLE** est utilisée pour créer une nouvelle table.
Voici le format d'une simple instruction de création d'une table (en pseudo-code) :

```
CREATE TABLE nom_de_la_table(
    colonne1 type_de_donnée,
    colonne2 type_de_donnée,
    colonne3 type_de_donnée
);
```

Voici le format de création d'une table si vous voulez utiliser des contraintes (en pseudo-code) :

```
CREATE TABLE nom_de_la_table(
    column1 type_de_donnée contrainte,
    column2 type_de_donnée contrainte,
    column2 type_de_donnée contrainte
);
```

Exemple de la création d'une table que nous appellerons employé:

```
CREATE TABLE employe(
    id INT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    prenom VARCHAR(25),
    nom VARCHAR(25),
    age INT(3),
    adresse VARCHAR(50),
    ville_id INT(5),
    CONSTRAINT fk_ville_id FOREIGN KEY (ville_id)
    REFERENCES ville(id)
);
```

Si on regarde l'instruction on peut différencier:

- **Les noms de colonnes:**
  id, prenom, nom, age, adresse, ville_id
- **Les types de données:**
  VARCHAR, INT
- **Les contraintes:**
  NOT NULL, PRIMARY KEY , AUTO_INCREMENT, CONSTRAINT fk_ville_id FOREIGN KEY REFERENCES ville(id).
  Comprendre la contrainte FOREIGN KEY:
  Dans l'exemple, on crée une clé étrangère (FK ou Foreign Key) vers la table ville, concrètement elle est liée à la colonne id de cette dernière table. Le nom fk_ville_id est le nom que nous avons donné à notre contrainte. Ce n'est pas obligatoire mais c'est une bonne pratique d'en donner un.

2.7 Contraintes

Nous avons introduit le concept de contrainte dans le chapitre consacré à l'introduction du cours. Comme nous avons vu, les contraintes sont utilisées pour limiter le type de données qui peuvent être insérées dans une table. Ceci garantit l'exactitude et la fiabilité des données d'une table.
Si nous essayons de réaliser une action qui viole une contrainte, l'action est refusée.

Voici une liste des contraintes les plus utilisées dans SQL :

- **NOT NULL** - Assure qu'une colonne ne peut pas avoir une valeur NULL
- **UNIQUE** - Assure que toutes les valeurs d'une colonne sont différentes.

- **PRIMARY KEY** - Une combinaison de NOT NULL et UNIQUE. Identifie de façon unique chaque ligne d'une table.
- **FOREIGN KEY** - Sert à relier deux tables entre elles.
- **CHECK** - Assure que toutes les valeurs d'une colonne satisfont une condition spécifique.
- **DEFAULT** - Définit une valeur par défaut pour une colonne lorsqu'aucune valeur n'est spécifiée.
- **INDEX** - Permet de créer et de récupérer très rapidement les données de la base de données.

Exemple de violation d'une contrainte
Imaginons que nous insérons notre premier employé dans notre table employe:

```
INSERT INTO employe(id, prenom, nom, adresse, ville)
VALUES (1, "Jean", "Dupont", "8 Avenue des Pins", 12);
```

Pour l'instant tout va bien. Maintenant, nous allons voir ce qui se passe si on essaie d'insérer un autre employé avec le même id:

```
INSERT INTO employe(id, prenom, nom, adresse, ville)
VALUES (1, "Marie", "scherbatsky", "12 rue du Moulin", 9);
```

Si vous essayez d'insérer à nouveau le même id, vous allez recevoir une erreur similaire à celle-ci:

> Duplicata du champ '1' pour la clef 'PRIMARY'

2.8 Créer / Supprimer une table ou une base de données

```
DROP TABLE employe;
--Si la table n'existe pas, la commande lance un erreur, donc on peut vérifier avant de supprimer la table
DROP TABLE if exists employe;
--Créer la table employe
CREATE DATABASE employe;
--Supprimer la base de données employe
DROP DATABASE employe;
--Si la base de données n'existe pas, la commande lance un erreur, donc on peut vérifier avant de supprimer la base de données
DROP DATABASE if exists employe;
```

2.9 Modification d'une table

```
--Ajouter une colonne
ALTER TABLE employe ADD country CHAR(22);
--Effacer la colonne nom
ALTER TABLE employe DROP COLUMN nom;
--Si lors d'une nouvelle insertion le nom n'est pas défini, il sera Martin par défaut
ALTER TABLE employe ALTER nom SET DEFAULT 'Martin';
```