SQL les langages de définition et de manipulation des données
A. Le LDD (create, alter, drop)

1. Création de la base : > SQL 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.

Syntaxe

Pour créer une base de données qui sera appelé "ma_base" :

```
CREATE DATABASE ma_base
```

Remarque : 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
```

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.

2. Suppression de la base :> SQL DROP DATABASE

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

```
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.

Remarque :

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
```

3. Création d'une table :> SQL CREATE TABLE

La commande CREATE TABLE permet de créer une table en SQL. La création d'une table sert à définir les colonnes et le type de données qui seront contenus dans chacun des colonnes (entier, chaîne de caractères, date, valeur binaire …).

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 )
```

Pour chaque colonne, il est également possible de définir des contraintes telles que :

- **NOT NULL :** empêche l'absence de valeur (null) pour une colonne.
- **DEFAULT :** attribue une valeur par défaut si aucune valeur n'est indiquée pour cette colonne lors de l'ajout d'une ligne.
- **PRIMARY KEY :** indique que cette colonne est la clé primaire.

Exemple

Créons une table utilisateur, dans laquelle chaque ligne correspond à un utilisateur inscrit sur un site web.

```
CREATE TABLE  user
(
   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
)
```

Commentaires 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 chaine de 100 caractères au maximum
- **prenom :** idem mais pour le prénom
- **email :** adresse email (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

Dans le langage SQL la "PRIMARY KEY", autrement 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 clé primaire est un index, chacune des tables ne peut contenir qu'une seule clé primaire, composée d'une ou plusieurs colonnes.
Fréquemment, la clé est une colonne numérique qui s'incrémente automatiquement à chaque insertion de ligne grâce à AUTO_INCREMENT.

Syntaxe

```
CREATE TABLE `nom_de_la_table` (
  id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  [...]
);
```

Autre syntaxe possible :

```
CREATE TABLE `nom_de_la_table` (
  `id` INT NOT NULL AUTO_INCREMENT,
  [...],
  PRIMARY KEY (`id`)
);
```

Exemple 1

```
CREATE TABLE `utilisateur` (
`id` INT NOT NULL AUTO_INCREMENT,
`nom` VARCHAR(50),
`email` VARCHAR(50),
`date_inscription` DATE,
PRIMARY KEY (`id`)
);
```

4. Modification du schéma de la table : > SQL ALTER TABLE

La commande ALTER TABLE en SQL permet de modifier une table existante en ajoutant, modifiant ou supprimant une colonne,

Syntaxe :

```
ALTER TABLE nom_table
  <détail-instruction>
```

4.1 Ajout d'une colonne

```
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)
```

4.2 Suppression d'une colonne

Il y a 2 manières totalement équivalente pour supprimer une colonne:

```
ALTER TABLE nom_table
DROP nom_colonne
```

Ou

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

4.3 Modification d'une colonne

il y a différentes syntaxes selon le SGBD.

- **Syntaxe MySQL et Oracle :**

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

- **Syntaxe PostgreSQL**

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

4.4 Renommer une colonne

- **Syntaxe 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
```

- **Syntaxe PostgreSQL**

Pour PostgreSQL le type n'est pas demandé

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

5.Suppression d'une table :> SQL 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.

Syntaxe

```
DROP TABLE nom_table
```

NB : s'il y a une dépendance 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, supprimer les tables filles avant de supprimer la table mère.

B. Le LMD (insert, update, delete)

1. Insertion de ligne : > INSERT INTO

1.1. Insertion d'une ligne à la fois

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 qu'on souhaite remplir. Il est possible d'insérer une ligne renseignant seulement une partie des colonnes

a) Insérer une ligne en spécifiant toutes les colonnes

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

b) Insérer une ligne en spécifiant seulement les colonnes souhaitées

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

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

c) Insertion de plusieurs lignes à la fois

```
INSERT INTO client (prenom, nom, ville, age)
 VALUES
('med', 'aloui', 'mila', 24),
('salim', 'kara', 'constantine', 36),…… ;
```

d) La commande 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.

Syntaxe

Cette commande s'effectue au sein de la requête INSERT INTO avec la syntaxe suivante:

```
INSERT INTO table (a, b, c)
VALUES (1, 20, 68)
ON DUPLICATE KEY UPDATE a=a+1
```

A noter : cette requête se traduit comme suit :

1. insérer les données **a, b et c** avec les données respectives de **1, 20 et 68**
2. Si la clé primaire existe déjà pour ces valeurs alors seulement faire une mise à jour de a = a+1

Exemple avec la commande WHERE

Grâce à la commande "ON DUPLICATE KEY" Il est possible d'enregistrer la date à laquelle la données est insérée pour la première fois et la date de dernière mise à jour, comme le montre la commande ci-dessous:

```
INSERT INTO table (a, b, c, date_insert)
VALUES (1, 20, 1, NOW())
ON DUPLICATE KEY UPDATE date_update=NOW
WHERE c=1
```

A noter : cette requête se traduit comme suit :

1. insérer les données a, b, c et date_insert, avec les données respectives de 1, 20, 1 ainsi que la date et l'heure actuelle
2. Si la clé primaire existe déjà pour ces valeurs alors mettre a jour la date et l'heure du champ "date_update"
3. Effectuer la mise à jour uniquement sur les champs où c = 1

Exemple

Imaginons une application qui laisse les utilisateurs voter pour les produits qu'ils préfèrent. Le système de vote est très simple et est basé sur des **+1**. La table des votes contient le nombre de votes par produits avec la date du premier vote et la date du dernier vote.

Table vote :

id	produit_id	vote_count	vote_first_date	vote_last_date
1	46	2	2022-04-25 17:45:24	2023-02-16 09:47:02
2	39	4	2022-04-28 16:54:44	2023-02-14 21:04:35
3	49	1	2022-04-25 19:11:09	2023-01-06 20:32:57

Pour n'utiliser qu'une seule ligne qui permet d'ajouter des votes dans cette table, sans se préoccuper de savoir s'il faut faire un INSERT ou un UPDATE, il est possible d'utiliser la requête SQL suivante:

```
INSERT INTO vote (produit_id, vote_count, vote_first_date, vote_last_date)
VALUES (50, 1, NOW(), NOW())
ON DUPLICATE KEY UPDATE vote_count = vote_count+1, vote_last_date = NOW()
```

Dans cette requête la date et l'heure est générée automatiquement avec la fonction NOW().

Résultat après la première exécution de la requête:

id	produit_id	vote_count	vote_first_date	vote_last_date
1	46	2	2022-04-25	2023-02-16

id	produit_id	vote_count	vote_first_date	vote_last_date
			17:45:24	09:47:02
2	39	4	2022-04-28 16:54:44	2023-02-14 21:04:35
3	49	1	2022-04-25 19:11:09	2023-01-06 20:32:57
4	55	1	2023-04-02 15:06:34	2023-04-02 15:06:34

Ce résultat montre bien l'ajout d'une ligne en fin de table, donc la requête a été utilisé sous la forme d'un INSERT. Après une deuxième exécution de cette même requête le lendemain, les données seront celles-ci:

id	produit_id	vote_count	vote_first_date	vote_last_date
1	46	2	2022-04-25 17:45:24	2023-02-16 09:47:02
2	39	4	2022-04-28 16:54:44	2023-02-14 21:04:35
3	49	1	2022-04-25 19:11:09	2023-01-06 20:32:57
4	55	2	2023-04-02 15:06:34	2023-04-03 08:14:57

Ces résultats montrent bien qu'il y a eu un vote supplémentaire et que la date du dernier vote a été mise à jour.

2. Modification de lignes :> 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

```
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.

NB : si la condition WHERE n'est pas utilisée, la même valeur est attribuée à la colonne nom_colonne_1 pour toutes les lignes d'une table

A noter, pour spécifier en une seule fois plusieurs modification, il faut séparer les attributions de valeur par des virgules :

```
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	Rue	ville	code_postal	pays
1	kara	Rue benmhidi	mila	43000	Algérie
2	aloui	Rue elkods	mila	43000	Algérie

a) Modifier une ligne

Pour modifier l'adresse du client kara :

```
UPDATE client
SET rue = 'rue 1er novembre',
  ville = 'constantine',
  code_postal = '25000'
WHERE id = 1 ;
```

Résultats :

id	nom	Rue	ville	code_postal	pays
1	kara	Rue 1er novembre	constantine	25000	Algérie
2	aloui	Rue elkods	mila	43000	Algérie

b) Modifier toutes les lignes

```
UPDATE client
SET pays = 'Algeria'
```

Résultats :

id	nom	Rue	ville	code_postal	pays
1	kara	Rue 1er novembre	constantine	25000	Algeria
2	aloui	Rue elkods	mila	43000	Algeria

3. Suppression de lignes : > DELETE

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

NB : 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

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

NB : 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.

a) 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	Rue	ville	code_postal	pays
2	aloui	Rue elkods	mila	43000	Algeria

b) Supprimer plusieurs lignes

Si l'ont souhaite supprimer les utilisateurs qui se sont inscrit avant le **10/04/2012**, il va falloir effectuer la requête suivante :

```
DELETE FROM `utilisateur`
WHERE condition
```

La requête permettra alors de supprimer les utilisateurs qui vérifient la condition

c) 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`
```

Ou la commande truncate

```
TRUNCATE TABLE `utilisateur`
```

Différence entre delete et truncate :
La commande TRUNCATE va ré-initialiser l'auto-incrémente s'il y en a un. Tandis que la commande DELETE ne ré-initialise pas l'auto-incrément.

4. Ajout/Mise a jour : > MERGE

Dans le langage SQL, la commande MERGE permet d'insérer ou de mettre à jour des données dans une table. Elle évite 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 standard : exemple

```
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 SGBDs 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

Vidage d'une table :> TRUNCATE TABLE

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.

Cette instruction diffère de la commande DROP qui à pour but de supprimer les données ainsi que la table qui les contient.

NB :

l'instruction TRUNCATE est semblable à l'instruction DELETE sans utilisation de WHERE.

Syntaxe

```
TRUNCATE TABLE `table`
```

Dans cet exemple, les données de la table "table" seront perdues une fois cette requête exécutée.

Exemple

Pour montrer un exemple concret de l'utilisation de cette commande, nous pouvons imaginer un système informatique contenant la liste des fournitures d'une entreprise. Ces données seraient tout simplement stockées dans une table "fourniture".

- **Table "fourniture" :**

Id	nom	date_ajout
1	Ordinateur	2023-04-05
2	Chaise	2023-04-14
3	Bureau	2023-07-18
4	Lampe	2023-09-27

- Il est possible de supprimer toutes les données de cette table en utilisant la requête suivante :
- `TRUNCATE TABLE `fourniture``
- Une fois la requête exécutée, la table ne contiendra plus aucun enregistrement. En d'autres mots, toutes les lignes de la table ci-dessus auront été supprimées.
- **Différence entre TRUNCATE et DELETE**
- La commande TRUNCATE s'avère être similaire à la commande DELETE, lorsqu'elle est utilisée de la façon suivante :
- `DELETE FROM `fourniture``
- Il y a toutefois une différence notable : la commande TRUNCATE va **ré-initialiser la valeur de l'auto-incrément**, s'il y en a un.