CREATION DE BASE DE DONNEES ET DES TABLES (SQL)

KONE Moussa Gbongué

Table des matières

Objectifs

À la fin de cette leçon, vous serez capable de:

- Créer une base de données et des tables
- Gérer les données dans une table

CRÉATION DE BASE DE DONNÉES ET DES TABLES

I

1. Création de base de données

1.1. Création

Une base de données informatique est un *ensemble de données* qui ont été stockées sur un support informatique, et *organisées et structurées* de manière à pouvoir facilement consulter et modifier leur contenu.

Remarque

NB : N'utilisez jamais, d'espaces ou d'accents dans vos noms de bases, tables ou colonnes. Au lieu d'avoir une colonne "date de naissance", préférez "date_de_naissance" ou "date_naissance". Et au lieu d'avoir une colonne "prénom", utilisez "prenom". Avouez que ça reste lisible, et ça vous évitera pas mal d'ennuis.

Évitez également d'utiliser des mots réservés comme nom de colonnes/tables/bases. Par "mot réservé", j'entends un mot-clé SQL, donc un mot qui sert à définir quelque chose dans le langage SQL

Nous allons donc créer notre base de données, que nous appellerons apprentissage. La commande SQL pour créer une base de données est la suivante :

```
CREATE DATABASE apprentissage ;
```

En créant la base de données, il faut également définir l'encodage qui sera utilisé. La commande SQL complète à taper pour créer votre base est :

```
CREATE DATABASE apprentissage CHARACTER SET 'utf8' ;
```

1.2. Suppression

Syntaxe

La commande SQL pour supprimerune base de données est :

```
DROP DATABASE IF EXISTS apprentissage ;
```

1.3. Utilisation d'une base de données

Syntaxe

Pour pouvoir agir sur cette base, vous devez d'abord la sélectionner. Ainsi, toutes les actions effectuées seront sur la base de données apprentissage (création et modification...) La commande SQL est la suivante :

```
USE apprentissage ;
```

2. Les types de données

Il est important de bien comprendre les usages et particularités de chaque type de données, afin de choisir le meilleur type possible lorsque vous définissez les colonnes de vos tables.

Types numériques

- *Nombres entiers*

TYPES	NOMBRE D'OCTETS	MIIMUM	MAXIMUM
TINYINT	1	-128	127
SMALLINT	2	32768	32767
MEDIUMINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

- *Nombres décimaux*

Cinq mots-clés permettent de stocker des nombres décimaux dans une colonne : *DECIMAL*, *NUMERIC*, *FLOAT* , *REAL* et *DOUBLE.*

Types alphanumériques

- *CHAR et VARCHAR*
  Pour stocker un texte relativement court (moins de 255 caractères), vous pouvez utiliser les types CHAR et VARCHAR.
- *TEXT*
  Il suffit alors d'utiliser le type TEXT, ou un de ses dérivés TINYTEXT, MEDIUMTEXT ou LONGTEXT.
- *TYPE BINAIRE*
  Comme les chaînes de type texte que l'on vient de voir, une chaîne binaire n'est rien d'autre qu'une suite de caractères.
  Cependant, si les textes sont affectés par l'encodage et l'interclassement, ce n'est pas le cas des chaînes binaires. Une chaîne binaire n'est rien d'autre qu'une suite d'octets. Aucune interprétation n'est faite sur ces octets. Ceci a deux conséquences principales.

  1. Une chaîne binaire traite directement l'octet, et pas le caractère que l'octet représente. Donc par exemple, une recherche sur une chaîne binaire sera toujours sensible à la casse, puisque "*A*" (*code binaire : 01000001*) sera toujours différent de "*a*" (*code binaire : 01100001*).
  2. Tous les caractères sont utilisables, y compris les fameux caractères de contrôle non-affichables définis dans la table ASCII.
     VARBINARY(x) et BINARY(x) permettent de stocker des chaînes binaires de xcaractères maximum (avec une gestion de la mémoire identique à VARCHAR(x) et CHAR(x)). Pour les chaînes plus longues, il existe les types TINYBLOB, BLOB, MEDIUMBLOB et LONGBLOB, également avec les mêmes limites de stockage que les types TEXT.

- *ENUM*
  ENUM est une colonne pour laquelle on définit un certain nombre de valeurs autorisées, de type "chaîne de caractère". Par exemple, si l'on définit une colonne statut (pour un type de statut d'étudiant) de la manière suivante :

```
statut ENUM('Orienté', 'Non orienté') ;
```

- *SET*
  SET est en effet une colonne qui permet de stocker une chaîne de caractères dont les valeurs possibles sont prédéfinies par l'utilisateur. La différence avec ENUM, c'est qu'on peut stocker dans la colonne entre 0 et x valeur(s), x étant le nombre de valeurs autorisées.

Types temporels

Les cinq types temporels de *MySQL* sont *DATE*, *DATETIME*, *TIME*, *TIMESTAMP* et *YEAR*.

- *DATE* : il est utilisé pour désigner les dates sous les formats ci-dessous :
  • 'AAAA-MM-JJ'
  • 'AAMMJJ'
  • 'AAAA/MM/JJ'
  • 'AA+MM+JJ'
  • 'AAAA%MM%JJ'
  • AAAAMMJJ (nombre)
  • AAMMJJ (nombre)
- *DATETIME* il est utilisé pour désigner les dates et heures sous les formats ci-dessous :
  • 'AAAA-MM-JJ HH:MM:SS'
  • 'AA*MM*JJ HH+MM+SS'
  • AAAAMMJJHHMMSS
- *TIME* : il est utilisé pour désigner l'heure sous les formats ci-dessous :
  • 'HH:MM:SS'
  • 'HHH:MM:SS'
  • 'MM:SS'
  • 'J HH:MM:SS'
  • 'HHMMSS'
  • HHMMSS (nombre)
- *YEAR* : il est utilisé pour désigner l'année sous le format AAAA.
  `Exemple de : 2018`

3. Rôle de l'index, de la clé primaire et de la clé étrangère

Définition : Index

Un *index* est une structure qui reprend la liste ordonnée des valeurs auxquelles il se rapporte.

Les index sont utilisés pour accélérer les requêtes (notamment les requêtes impliquant plusieurs tables, ou les requêtes de recherche), et sont indispensables à la création de clés, étrangères et primaires, qui permettent de garantir l'intégrité des données de la base

Définition : Clé primaire

La clé primaire d'une table est une contrainte d'unicité, composée d'une ou plusieurs colonnes, et qui permet d'identifier de manière unique chaque ligne de la table.

Examinons plus en détail cette définition.

- *Contrainte d'unicité :* ceci ressemble fort à un index UNIQUE.
- *Composée d'une ou plusieurs colonnes :* comme les index, les clés peuvent donc être composites.

- *Permet d'identifier chaque ligne de manière unique :* dans ce cas, une clé primaire ne peut pas être NULL.

Définition : Clé étrangère

Les clés étrangères, qui permettent de gérer des *relations entre plusieurs tables*, et garantissent la cohérence des données.

4. Création de tables

Identité

Imaginez que quelqu'un ait le même nom de famille que vous, le même prénom, soit né dans la même ville et ait la même taille. En dehors de la photo et de la signature, quelle sera la différence entre vos deux cartes d'identité ? Son numéro !

Suivant le même principe, on va donner à chaque animal un *numéro d'identité*. La colonne qu'on ajoutera s'appellera donc id, et il s'agira d'un INT, toujours positif donc UNSIGNED. Selon la taille de l'élevage (la taille actuelle mais aussi la taille qu'on imagine qu'il pourrait avoir dans le futur !), il peut être plus intéressant d'utiliser un SMALLINT, voire un MEDIUMINT. Comme il est peu probable que l'on dépasse les 65000 animaux, on utilisera SMALLINT.

Ce champ ne pourra bien sûr pas être *NULL*, sinon il perdrait toute son utilité.

Clé primaire

La clé primaire d'une table est une *contrainte d'unicité*, composée d'une ou plusieurs colonnes. La clé primaire d'une ligne *permet d'identifier de manière unique cette ligne dans la table*. En utilisant *id* comme la clé primaire de la table *Etudiant* on aura *PRIMARY KEY(id)* comme déclaration de la clé primaire.

Lorsque vous insérerez une nouvelle ligne dans la table, MySQL vérifiera que vous insérez bien un id, et que cet id n'existe pas encore dans la table. Si vous ne respectez pas ces deux contraintes, MySQL n'insérera pas la ligne et vous renverra une erreur.

Auto-incrémentation

Il faut donc, pour chaque animal, décider d'une valeur pour id. Le plus simple, et le plus logique, est de donner le numéro 1 au premier individu enregistré, puis le numéro 2 au second, etc. Mais si vous ne vous souvenez pas quel numéro vous avez utilisé en dernier, pour insérer un nouvel animal il faudra récupérer cette information dans la base, ensuite seulement vous pourrez ajouter une ligne en lui donnant comme *id* le dernier *id* utilisé + *1*. C'est bien sûr faisable, mais c'est fastidieux... Heureusement, il est possible de demander à *MySQL* de faire tout ça pour nous ! Comment ? En utilisant l'*auto-incrémentation* des colonnes. Incrémenter veut dire "ajouter une valeur fixée". Donc, si l'on déclare qu'une colonne doit s'auto-incrémenter (grâce au mot-clé *AUTO_INCREMENT*), plus besoin de chercher quelle valeur on va mettre dedans lors de la prochaine insertion. *MySQL* va chercher ça tout seul comme un grand en prenant la dernière valeur insérée et en l'incrémentant de 1.

Définition : Les moteurs de tables

Les moteurs de tables sont une spécificité de MySQL. Ce sont des moteurs de stockage. Cela permet de gérer différemment les tables selon l'utilité qu'on en a. Les deux moteurs les plus connus sont *MyISAM* et *InnoDB*. Préciser un moteur lors de la création de la table.

Pour qu'une table utilise le moteur de notre choix qui est par exemple InnoDB, il suffit d'ajouter ceci à la fin de la commande de création :

```
ENGINE =INNODB ;
```

Syntaxe

création de table

```
CREATE TABLE produit
(
Id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
Libelle VARCHAR(40) NOT NULL,
Description TEXT,
date_saisie DATETIME NOT NULL,
quantite INT,
PRIMARY KEY (id)
)
ENGINE=INNODB;
```

Syntaxe : Suppression d'une table

La commande pour supprimer une table est la même que celle pour supprimer une base de données. Elle est, bien sûr, à utiliser avec prudence, car irréversible :

```
DROP TABLE IF EXISTS produit;
```

5. Gestion des contraintes d'intégrité

Avec MySQL, les tables pourront gérer les contraintes d'intégrité uniquement si elles sont de type INNODB (ce qui n'est généralement pas le cas par défaut). Pour vous assurer de bien créer des tables de type INNODB vous devez ajouter, à la fin de la requête de création de la table, l'instruction TYPE=INNODB.

```
CREATE TABLE ... ENGINE=INNODB ;
```

Syntaxe : Ajout de la contrainte

L'ajout de la contrainte se fait via l'instruction *FOREIGN KEY* suivi de la liste des champs liés (de la table à laquelle s'applique l'instruction) suivi de l'instruction *REFERENCES*, le nom de la table référencée et enfin la liste des champs référencés

```
... FOREIGN KEY (realisateur_id) REFERENCES realisateur(id) ...
```

Exemple

Si l'on reprend l'exemple initial des 2 tables liées film et realisateur. Les requêtes de création de ces tables sans contrainte d'intégrité seraient quelque chose du genre:

```
CREATE TABLE realisateur(
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
nom VARCHAR(40) NOT NULL,
prenoms VARCHAR(80)
) ENGINE=INNODB;

CREATE TABLE film(
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
titre VARCHAR(100) NOT NULL,
realisateur_id INT
) ENGINE=INNODB ;
```

Attention

Avec une contrainte d'intégrité les requêtes deviennent :

```
CREATE TABLE realisateur(
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
nom VARCHAR(40) NOT NULL,
prenoms VARCHAR(80)
) ENGINE=INNODB;

CREATE TABLE film(
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
tire VARCHAR(100) NOT NULL,
realisateur_id INT,
FOREIGN KEY (realisateur_id) REFERENCES realisateur (id)
) ENGINE=INNODB ;
```

6. Exercice

Pour la mise en place d'un répertoire téléphonique des étudiants de l'UVCI, M. GOVOU veut constituer une base de données pour faciliter sa tâche.

1. Identifier les différentes tables possibles
2. Créer une base de données bd_repertoire
3. Créer une table client (code_clt, nom_clt, adresse_clt, telephone_clt) avec le moteur INNODB
4. Créer une table facture (num_facture, date_facture, montant_facture, code_clt) avec le moteur INNODB
5. Utiliser la contrainte d'intégrité pour les clés étrangères

MANIPULATION DES DONNEES

II

Insertion de données

Deux possibilités s'offrent à nous lorsque l'on veut insérer une ligne (un enregistrement) dans une table : soit donner une valeur pour chaque colonne de la ligne, soit ne donner les valeurs que de certaines colonnes, auquel cas il faut bien sûr préciser de quelles colonnes il s'agit.

Nous avons la table *etudiant* avec les propriétés suivantes : matricule, nom, prenoms, date_naissance, sexe)

Syntaxe

Insertion sans préciser les colonne

```
INSERT INTO etudiant VALUES ('25K58B', 'KOFFI', 'Jean Francky', '2000-02-24', 'M') ;
```

Syntaxe

Insertion en précisant les colonnes

```
INSERT INTO etudiant(matricule, nom, prenoms, date_naissance, sexe)
VALUES('25K58B', 'KOFFI', 'Jean Francky', '2000-02-24', 'M') ;
```

Syntaxe

Insertion multiple

```
INSERT INTO etudiantVALUES ('25K58B', 'KOFFI', 'Jean Francky', '2000-02-24', 'M'),
                           ('25K58B', 'KOFFI', 'NULL', '2000-02-24', 'M'),
                           ('25K58B', 'KOFFI', 'Jean Francky', '2000-02-24', 'NULL') ;
```

Sélection de données

La requête qui permet de sélectionner et afficher des données s'appelle SELECT. SELECT permet donc d'afficher des données directement. Des chaînes de caractères, des résultats de calculs, etc.

Exemple

```
SELECT matricule, nom, prenoms, date_naissance, sexe FROM etudiant ;

Ou

SELECT * FROM etudiant ;
```

Fondamental

(*)Signifie : Sélectionner toutes les colonnes de la table *etudiant*.

Syntaxe : La clause WHERE

La clause WHERE ("*où*" en français) permet de restreindre les résultats selon des critères de recherche.

```
SELECT * FROM etudiant WHERE matricule = 'MD250';//Un étudiant selon son matricule

SELECT * FROM etudiant WHERE date_naissance < '2001-05-20';//Etudiants nés avant 2001

SELECT * FROM etudiant WHERE nom <> 'KONE';//Tous les Etudiants dont le nom n'est pas KONE

SELECT * FROM etudiant WHERE nom= 'DJOMAN' AND date_naissance < '2001-05-20';//Etudiants dont
le nom est DJOMAN et qui sont nés avant 2001

SELECT * FROM etudiant WHERE nom= 'DJOMAN' OR nom = 'ZADI';//Etudiants dont le nom est
DJOMAN ou le nom est ZADI

SELECT * FROM etudiant ORDER BY nom ASC;//Tous les Etudiants par ordre alphabétique

SELECT * FROM etudiant WHERE sexe='M'ORDER BY nom ASC;//Tous les Etudiants du sexe masculin par
ordre alphabétique

SELECT DISTINCT nom FROM etudiant;// Tous les étudiants sans doublons sur le nom

SELECT * FROM etudiant LIMIT 10;//Tous les 10 premiers Etudiants
```

Les opérateurs de comparaison

Les opérateurs de comparaison sont les symboles que l'ont utilise pour définir les critères de recherche. Huit opérateurs simples peuvent être utilisés

Opérateur	Signification
=	Égal
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
<> où ! =	Différent
<=>	Égal (valable pour NULL aussi)

Syntaxe

```
SELECT * FROM etudiant WHERE matricule = 'MD250';//Un étudiant selon son matricule
SELECT * FROM etudiant WHERE date_naissance < '2001-05-20';//Etudiants nés avant 2001
SELECT * FROM etudiant WHERE nom <> 'KONE';//Tous les Etudiants dont le nom n'est pas KONE
SELECT * FROM etudiant WHERE nom= 'DJOMAN' AND date_naissance < '2001-05-20';//Etudiants dont le nom est DJOMAN et qui sont nés avant 2001
```

Combinaisons de critères

Ils sont au nombre de 4 opérateurs et sont utilisés pour de combiner des critères.

Opérateur	Symbole	Signification
AND	&&	ET
OR	||	OU
XOR		OU exclusif
NOT	!	NON

Syntaxe

```
SELECT * FROM etudiant WHERE nom= 'KONAN' AND date_naissance < '1997-01-05'
SELECT * FROM etudiant WHERE nom= 'YEO' OR nom= 'KONE'
```

Suppression de données

La commande utilisée pour supprimer des données est *DELETE*. Cette opération est irréversible, soyez très prudents !

On utilise la clause *WHERE* de la même manière qu'avec la commande *SELECT* pour préciser quelles lignes doivent être supprimées.

Syntaxe

```
DELETE FROM etudiant ;    //Supprime tous les s Etudiants

DELETE FROM etudiant WHERE matricule='LK87P' ;    //Supprime l'étudiant selon le matricule
```

Modification de données

La modification des données se fait grâce à la commande UPDATE, dont la syntaxe est la suivante :

Syntaxe

```
UPDATE etudiant SET nom='DAGO', prenoms='Anicet' WHERE matricule='LK87P' ;
//Modifie le nom et le prénom de l'étudiant ayant le matricule LK87P

UPDATE etudiant SET sexe='M'; //Modifie toute la liste des étudiants en leur donnant le sexe M
l'étudiant  selon le matricule
```

1. Exercice

Soient les tables *t_client* et *t_compte* ci-dessous :

t_client (*id_client*, nom, prenoms, sexe, contact, reseau_mobile) ;

t_compte (*id_compte*, numero_compte, solde, date_creation, #id_client) ;

1. Ecrire le code SQL pour insérer trois (03) clients dans la table.
2. Exemple : nom= 'KOUADIO', prenoms='Sosthène', sexe='M', contact='07 08 01 00', reseau_mobile='Orange'
3. Ecrire le code SQL, pour afficher la liste de tous les clients par ordre alphabétique.
4. Ecrire le code SQL, pour afficher tous les clients qui sont sur le réseau mobile Orange.
5. Ecrire le code SQL, pour afficher le nom, prénoms et sexe de tous les hommes qui sont sur le réseau MTN.
6. Ecrire le code SQL, pour afficher les informations du compte d'un client selon son id_client.
7. Ecrire le code SQL, pour afficher les 5 premiers clients enregistrés dans la journée en cours.
8. Ecrire le code SQL, pour afficher tous les comptes qui ne sont pas vides.
9. Ecrire le code SQL, pour afficher tous les comptes dont le solde est 0, et qui ont été créés dans le mois de Juin.
10. Ecrire le code SQL, pour afficher le numéro de compte, nom, prénoms et contact des clients qui sont sur le réseau Moov, et dont le solde est supérieur à 50.000F