Bases de données : Langage SQL

1. Introduction

SQL (**S**tructured **Q**uery **L**anguage) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. Créé en 1974, normalisé depuis 1986, le langage **SQL** est reconnu par la grande majorité des systèmes de gestion de bases de données relationnelles (abrégé SGBDR) du marché.

Les instructions SQL s'écrivent d'une manière qui ressemble à celle de phrases ordinaires en anglais. Cette ressemblance voulue vise à faciliter l'apprentissage et la lecture.

Les instructions SQL couvrent quatre domaines :

- Langage de définition de données (**LDD**), désigne les commandes SQL utilisées pour décrire les objets d'une base de données : tables, relations, index, vues, fonctions, procédures, déclencheurs, etc. ;
- Langage de manipulation de données (**LMD**), représente les commandes SQL employées dans la recherche, l'ajout, la modification ou la suppression des données ;
- Langage de contrôle de données (**LCD**), incarne les commandes SQL renforçant la sécurité des données en limitant l'accès aux données aux entités autorisées ;
- Langage de contrôle des transactions (**LCT**),

2. Langage de définition des données

Les commandes SQL mentionnées dans cette partie concernent les logiciels MySQL et MariaDB.

A. Création d'une base de données

```
CREATE DATABASE nom_base;
```

B. Suppression d'une base de données

```
DROP DATABASE nom_base;
```

C. Création d'une table

Une table est créée à l'aide de la commande **`CREATE TABLE`**. Cette commande possède le format suivant :

```
CREATE TABLE nom_table (
  définition_colonne1,
  …,
  définition_colonnen,
  définition_contrainte1,
  …,
  définition_contraintep,
);
```

La clause "**`définition_colonne`**$_i$" permet de préciser les caractéristiques d'une colonne. Elle a la syntaxe suivante :

```
nom_colonne type [[NOT] NULL] [DEFAULT valeur] [contrainte_colonne]
```

Le **type** d'une colonne peut être :

Type	Syntaxe	Description
Alphanumérique	**VARCHAR(n)** **CHAR(n)**	Une chaîne de caractères de n caractères maximum
	TEXT	Une chaîne de caractère de taille ≤ 65535 caractères
Date / Heure	**DATE**	La date au format : 'yyyy-mm-dd'
	TIME	L'heure au format : 'HH:MM:SS'
	DATETIME	La date et l'heure au format : 'yyyy-mm-dd HH:MM:SS'
Entier	**INT**	Un nombre entier dans l'intervalle ± 2 147 483 647
Réel	**FLOAT** **DOUBLE**	Un nombre réel à virgule flottante
	NUMERIC(n, m) **DECIMAL(n, m)**	Un nombre réel à n chiffres dont m décimales

L'option **[[NOT] NULL]** indique si la colonne est obligatoire ou optionnelle.

L'option **[DEFAULT valeur]** permet d'attribuer une valeur par défaut à une colonne lorsqu'aucune valeur ne lui est affectée.

L'option **[contrainte_colonne]** permet de préciser une contrainte d'intégrité relative à la colonne. Elle possède la syntaxe suivante :

```
[CONSTRAINT contrainte]
{ PRIMARY KEY
| REFERENCES nom_table(nom_colonne)
    [ON UPDATE CASCADE]
    [ON DELETE CASCADE]
| CHECK (condition)}
```

Le mot clé **[CONSTRAINT contrainte]** est optionnel, il sert juste à attribuer un nom à la contrainte. Une contrainte peut être composée par la combinaison d'une ou de plusieurs options :

- **PRIMARY KEY** : la colonne est une clé primaire;
- **REFERENCES nom_table(nom_colonne)** : la colonne est une clé étrangère qui relie cette colonne avec la colonne "**nom_colonne**" de la table mère "**nom_table**".
  - **[ON UPDATE CASCADE]** et **[ON DELETE CASCADE]** sont utilisés pour maintenir l'intégrité référentielle
- **CHECK(condition)** : Les valeurs insérées dans cette la colonne doivent vérifier la **condition** Mentionnée.

Exemples : En utilisant la première syntaxe de création des tables,

```
CREATE TABLE article (
  CodeA VARCHAR(20) PRIMARY KEY,
  DesA VARCHAR(50) NOT NULL,
  PU DECIMAL(8, 3) CHECK(PU > 0),
  Qte INT DEFAULT 0 CHECK(Qte >= 0)
);
```

```
CREATE TABLE commande (
  numCmd VARCHAR(20)
    CONSTRAINT pk_cli PRIMARY KEY,
  dateCmd DATE NOT NULL,
  codeCl VARCHAR(20)
    REFERENCES client(codecl)
);
```

La clause "**définition_contrainte$_i$**" sert à définir une contrainte d'intégrité au niveau de la table. Elle doit être utilisée lorsqu'une contrainte s'applique sur plusieurs colonnes. Sa syntaxe est comme suit :

```
[CONSTRAINT contrainte]
{ PRIMARY KEY (colonne1, colonne2,...)
| FOREIGN KEY (colonne1, colonne2,...)
    REFERENCES nom_table[(col1,col2,...)]
      [ON UPDATE CASCADE]
      [ON DELETE CASCADE]
|CHECK (condition)}
```

Le mot clé **[CONSTRAINT contrainte]** est optionnel, il sert juste à attribuer un nom à la contrainte. Les options **PRIMARY KEY** et **CHECK** possèdent la même signification avancée précédemment.

L'option **FOREIGN KEY (colonne$_1$, colonne$_2$, ...) REFERENCES nom_table(col$_1$, col$_2$,...) [ON UPDATE CASCADE] [ON DELETE CASCADE]** permet de définir les colonnes "**colonne$_1$, colonne$_2$, ...**" comme une clé étrangère qui référence les colonnes "**col$_1$, col$_2$, ...**" de la table **nom_table**.

Exemple : En utilisant la seconde syntaxe de création des tables,

```
CREATE TABLE details_commande (
  NumCmd VARCHAR(20),
  NumLigne INT,
  QteCmd INT,
  PRIMARY KEY (NumCmd, NumLigne),
  FOREIGN KEY (NumCmd) REFERENCES commande(NumCmd),
  CHECK(QteCmd > 0)
);
```

D. Suppression d'une table

Pour supprimer le contenu et la structure d'une table in utilise la commande :

```
DROP TABLE nom_table;
```

E. Modification de la structure d'une table

La structure d'une table est modifiée à l'aide la commande **ALTER TABLE**. Cette commande est utile pour ajouter, supprimer, et/ou modifier les colonnes et les contraintes d'intégrité.

i. Renommer une table

```
ALTER TABLE table_name
  RENAME TO new_table_name;
```

ii. Ajouter des colonnes

Une seule colonne

```
ALTER TABLE table_name
  ADD [COLUMN] col_name col_def [FIRST | AFTER col_name];
```

Plusieurs colonnes

```
ALTER TABLE table_name
  ADD [COLUMN] (col_name col_def,  …);
```

iii. Modifier la définition d'une colonne

Sans changer le nom de la colonne.

```
ALTER TABLE table_name
  MODIFY [COLUMN] col_name col_def [FIRST | AFTER col_name];
```

Changer le nom de la colonne.

```
ALTER TABLE table_name
  CHANGE [COLUMN] old_col_name col_name col_def [FIRST | AFTER col_name];
```

Changer le nom d'une colonne (**MariaDB version 10.5.2 ou plus**)

```
ALTER TABLE table_name
  RENAME [COLUMN] old_col_name TO col_name;
```

iv. Supprimer une colonne

```
ALTER TABLE table_name
  DROP [COLUMN] col_name;
```

v. Définir une clé primaire

```
ALTER TABLE table_name
  ADD [CONSTRAINT [symbol]] PRIMARY KEY (col_name, …);
```

vi. Définir une contrainte d'unicité

```
ALTER TABLE table_name
  ADD [CONSTRAINT [symbol]] UNIQUE (col_name, …);
```

vii. Définir une clé étrangère

```
ALTER TABLE table_name
  ADD [CONSTRAINT [symbol]]
    FOREIGN KEY (col_name, …) REFERENCES table_name(col_name, …)
    [ON DELETE CASCADE] [ON UPDATE CASCADE];
```

Définir une contrainte de domaine

```
ALTER TABLE table_name
  ADD [CONSTRAINT [symbol]] CHECK(constraint_def);
```

viii. Supprimer des contraintes

Clé primaire

```
ALTER TABLE table_name
  DROP PRIMARY KEY;
```

Clé étrangère

```
ALTER TABLE table_name
  DROP FOREIGN KEY fk_symbol;
```

Une contrainte quelconque

```
ALTER TABLE table_name
  DROP CONSTRAINT constraint_name;
```

Exemple :

Soit la base de données "**films**" ayant la représentation graphique incomplète suivante :

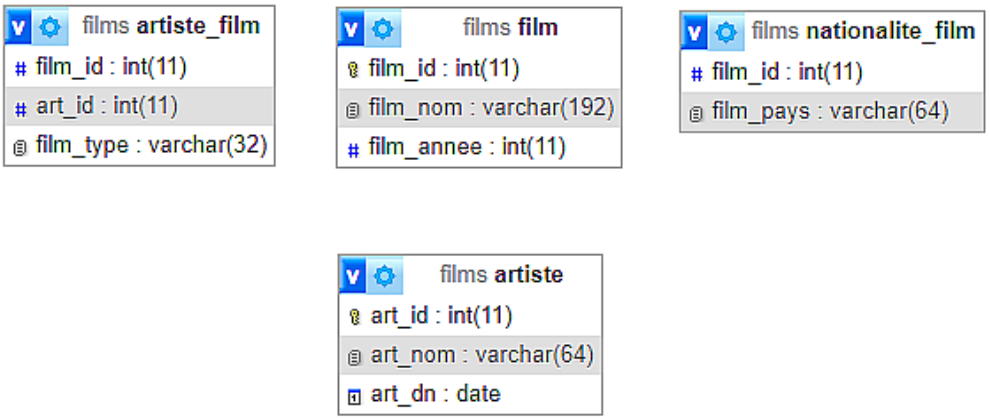

*Figure 1: Représentation graphique incomplète de la base de données **films***

On veut écrire des requêtes SQL afin de :

1. Ajouter le champ "**film_genre**" à la table **film** pour indiquer le genre d'un film : policier, aventure, romantique, etc.
2. Renommer le champ de la table **film** de "**film_nom**" en "**film_titre**" tout en conservant le même type et la même taille de ce champ,
3. Réduire la taille du champ "**film_titre**" de la table **film** de 192 caractères à 128 caractères,
4. Définir les champs "**film_id**" et "**art_id**" comme la clé primaire de la table **artiste_film**,
5. Définir les champs"**film_id**" et "**art_id**" de la table **artiste_film** comme étant des clés étrangères,

```
ALTER TABLE film
    -- (r1) Ajouter un champ à film
    ADD COLUMN film_genre VARCHAR(64) NOT NULL,

    -- (r2) Renommer un champ
    CHANGE COLUMN film_nom film_titre VARCHAR(192);

ALTER TABLE film
    -- (r3) Changer la taille d'une colonne
    MODIFY film_titre VARCHAR(128);

ALTER TABLE artiste_film
    -- (r4) Ajouter la clé primaire
    ADD CONSTRAINT pk_art_film PRIMARY KEY (film_id, art_id),

    -- (r5) Ajouter les clés étrangères
    ADD CONSTRAINT fk_art_film1 FOREIGN KEY (film_id)
        REFERENCES film(film_id),
    ADD CONSTRAINT fk_art_film2 FOREIGN KEY (art_id)
        REFERENCES artiste(art_id);
```

3. Langage de Manipulation des Données

Les exemples montrés dans ce cours utilisent la base de données "films" dont la représentation graphique finale est montrée en figure 2 :

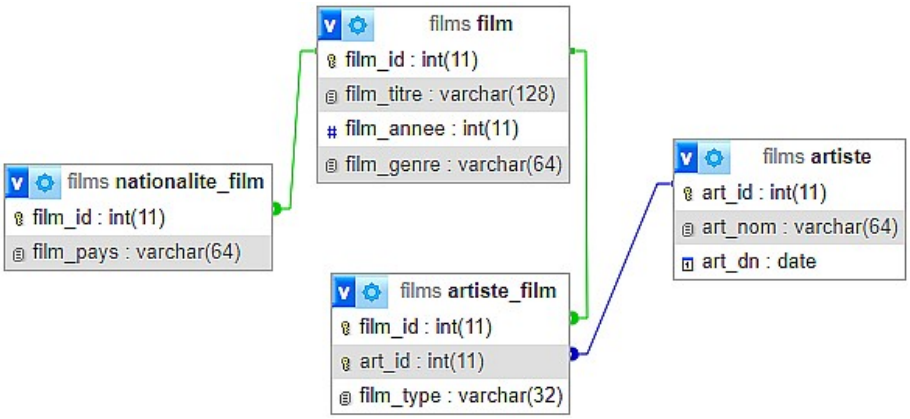

*Figure 2: Représentation graphique de la base de données **films***

A. Insertion de données

Pour insérer de nouvelles lignes dans une table on utilise la commande **`INSERT`**.

```
INSERT INTO nom_table [liste_nom_colonnes]
VALUES (liste des valeurs);
```

La "**`liste_nom_colonnes`**" est la liste des colonnes dans lesquelles se fera l'opération d'insertion. Cette liste peut être omise dans le cas où l'insertion concerne tous les champs de la table dans leur ordre de création.

Exemples : Insertion de quelques artistes dans la table artiste

```
-- Forme 1 : Les noms des champs à insérer sont explicités
INSERT INTO artiste (art_id, art_nom, art_dn)
VALUES (1, 'Pascal Obispo', '1965-01-06');

-- Forme 2 : Les noms des champs sont omis
INSERT INTO artiste
VALUES (2, 'Dalida', '1933-01-17');

-- Forme 3 : Uniquement les noms de quelques champs sont mentionnés
INSERT INTO artiste (art_nom, art_dn)
VALUES ('Robbie Williams', '1974-02-13');
```

B. Suppression de données

La commande SQL **`DELETE`** est utilisée pour supprimer des lignes d'une table.

```
DELETE FROM nom_table [WHERE condition]
```

La clause "**`[WHERE condition]`**" est optionnelle. Le paramètre "**`condition`**" dans cette clause indique qu'uniquement les lignes vérifiant la condition seront supprimées.

Si la clause "**`[WHERE condition]`**" est absente, dans ce cas, toutes les lignes de la table "**`nom_table`**" seront supprimées.

Attention : La suppression d'une ligne appartenant à la table "**nom_table**" peut entraîner la suppression d'autres lignes appartenant à d'autres tables lorsqu'il existe une contrainte d'intégrité référentielle de suppression en cascade.

Exemple : Dans le but d'alléger la table **artiste** on veut supprimer tous les artistes nés avant les années 1950.

```
DELETE FROM artiste WHERE art_dn <= '1950-01-01';
```

C. Mise à jour des données

La commande SQL **UPDATE** est utile pour modifier le contenu d'une table.

```
UPDATE nom_table
SET
  nom_colonne1 = expression1,
  …,
  nom_colonnen = expressionn
[WHERE condition]
```

Toutes les lignes vérifiant la "**condition**" mentionnée dans la clause "**[WHERE condition]**" seront mises à jour. Si cette dernière est omise toutes les lignes de la table seront modifiées.

Exemple : On veut corriger l'année de sortie du film "**Matrix**".

```
UPDATE film SET film_annee = 1999 WHERE film_titre = 'Matrix';
```

D. Recherche de données

La recherche des données peut :

- concerner ***quelques colonnes*** d'une table : on parle, alors, de **Projection**,
- se rapporter à ***quelques lignes*** d'une table : il s'agit dans ce cas d'une **Sélection**,
- toucher ***deux tables en relation*** : il est question alors de **Jointure**,
- être une combinaison de ces trois actions : **Projection**, **Sélection** et **Jointure**.

La recherche se fait à l'aide de la commande SQL **SELECT**.

Sa syntaxe complète est la suivante :

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
[FROM table_references]
[WHERE where_condition]
[GROUP BY {col_name | expr | position}, ...]
[HAVING where_condition]
[ORDER BY {col_name | expr | position} [ASC | DESC], ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

L'opération de recherche fera l'objet d'un autre cours plus détaillé.