

Le langage SQL (*Structured Query Language*, langage de requête structuré) est un langage utilisé dans beaucoup de SGBD (Système de Gestion de Base de Données). Il a été développé par IBM en 1970 et inspiré du modèle relationnel que l'on a décrit dans le chapitre précédent. Depuis, il a été recommandé par l'ANSI (*American National Standard Institute*) et adopté comme norme internationale par l'ISO (*International Organization for Standardization*) sous l'appellation ISO-9075.

Les SGBD (libres ou propriétaires) basés sur le langage SQL suivent en partie cette norme et s'il est parfois possible que certaines instructions diffèrent d'un SGBD à l'autre, la structure du langage reste toujours la même. On utilisera le SGBD MariaDB qui est un dérivé du SGBD MySQL utilisé sur les distributions GNU/Linux.

Le langage SQL permet deux types d'actions : les **mises à jour** (création de relations, ajout de données, suppression) et les **requêtes** (affichage). Ce chapitre traite des mises à jour.

## 1 Installation de MariaDB / MySQL sur le Raspberry

S'assurer que le Raspberry est connecté à internet, puis ouvrir un terminal et entrer à la suite les commandes suivantes :

```
1 sudo apt-get update
2 sudo apt-get install mariadb-server
3 sudo mysql_secure_installation
```

La dernière commande permet de configurer le logiciel (répondre oui à toutes les questions et définir un mot de passe). Le SGBD MariaDB est maintenant installé sur le Raspberry et un superutilisateur (appelé **root**) a été configuré.

```
1 sudo mysql -u root -p
```

On est maintenant connecté à MariaDB en tant que **root**.

```
1 MariaDB [(none)]> CREATE USER 'nom_utilisateur'@'localhost' IDENTIFIED BY 'mdp_utilisateur' ;
```

On vient de créer un autre utilisateur (mettre votre nom) qui pourra se connecter à partir du Raspberry (@'localhost')

```
1 MariaDB [(none)]> CREATE DATABASE bibliotheque ;
```

On vient de créer une nouvelle base de données nommée **bibliotheque**.

```
1 MariaDB [(none)]> GRANT ALL PRIVILEGES ON nom_base_de_donnees.* TO 'nom_utilisateur'@'localhost' ;
```

On vient d'accorder au nouvel utilisateur tous les droits sur la base de données **bibliotheque**.

```
1 MariaDB [(none)]> SELECT User, Host FROM mysql.user ;
```

On vérifie que tout s'est bien passé : doit s'afficher un tableau contenant **root** et l'autre utilisateur en **localhost**.

```
1 MariaDB [(none)]> quit ;
```

On vient de quitter MariaDB

```
1 mysql -u nom_utilisateur -p
```

On est maintenant connecté en tant que le nouvel utilisateur.

```
1 MariaDB [(none)]> USE bibliotheque ;
```

On a sélectionné la base de données.

### Quelques remarques :

- On sait qu'on est connecté à MariaDB grâce au `MariaDB [...]>` affiché dans le terminal.
- Chaque commande SQL se termine par `;` (après un an de Python, attention de ne pas l'oublier!).
- L'utilisateur **root** peut gérer les autorisations qu'ont les autres utilisateurs sur les différentes bases de données.
- Chaque utilisateur se connecte au SGBD à partir d'une certaine machine toujours précisée. Pour l'instant, cette machine est le Raspberry désigné par **localhost**.
- La commande **SELECT** permet d'afficher des tableaux, elle correspond au **print** de Python.

Toutes les commandes que l'on verra à partir de maintenant pourront être lancées au choix :

- du terminal Linux, en étant connecté en tant qu'utilisateur autorisé de la base de données ;
- d'une interface SQL ;
- à partir d'un fichier lancé avec la commande :

```
mysql -u nom_utilisateur -pmot_de_passe nom_de_la_base < fichier.sql
```

## 2 Création d'une table, un exemple

Dans le langage SQL, les termes peuvent différer de ce que l'on a vu dans le chapitre précédent. Ainsi, une relation est appelée en SQL une table et un attribut une colonne. Cela provient du fait que les relations y sont vues et représentées sous forme de tableaux.

Entrer les commandes suivantes :

```
1 CREATE TABLE Usager(nom VARCHAR(90), prenom VARCHAR(90), adresse VARCHAR(300), cp VARCHAR(5), ville
  VARCHAR(60), code CHAR(6) PRIMARY KEY) ;

1 CREATE TABLE Livre(titre VARCHAR(300), auteur VARCHAR(90), editeur VARCHAR(90), annee INT, isbn CHAR(17),
  num INT PRIMARY KEY) ;

1 CREATE TABLE Emprunt(code CHAR(15) REFERENCES Usager(code), num INT REFERENCES Livre(num) PRIMARY KEY,
  date_emprunt DATE) ;
```

On reconnaît à travers ces commandes les schémas des relations de notre exemple du chapitre BD1.

**Création de table :** Pour créer une table en SQL, la syntaxe est la suivante :

```
CREATE TABLE Nom_table(att1 DOM1, att2 DOM2, ...) ;
```

On y précise de plus les contraintes d'entités comme nous allons le voir ensuite.

## 3 Spécification des domaines

Le domaine de chaque attribut est spécifié à la création de la table. Décrivons les types disponibles en SQL :

**Types numériques :** Pour représenter les nombres, on dispose de plusieurs types à utiliser en fonction de la situation :

| Domaine          | Exact ou approché | Description   |
|------------------|-------------------|---|
| SMALLINT         | exact             | entier signé 16 bits                                |
| INTEGER          | exact             | entier signé 32 bits                                |
| INT              | exact             | alias pour INTEGER                                  |
| BIGINT           | exact             | entier signé 64 bits                                |
| DECIMAL(t,f)     | exact             | décimal signé de t chiffres dont f après la virgule |
| REAL             | approché          | flottant 32 bits                                    |
| DOUBLE PRECISION | approché          | flottant 64 bits                                    |

**Types textuels :** Pour représenter textes, on dispose de nombreux types à choisir en fonction de la situation. En voici quelques uns :

| Domaine    | Description                                 |
|------------|---|
| CHAR(n)    | chaîne de caractères de taille exactement n |
| VARCHAR(n) | chaîne de caractères de taille au plus n    |
| TEXT       | chaîne de caractères de taille quelconque   |

**Type booléen :** Pour représenter les booléens, on dispose du type BOOLEAN mais ce type est absent de certains SGBD. Dans ceux là, on pourra utiliser un SMALLINT (0 = faux, autre = vrai) ou un CHAR(1) contenant T ou F.

**Types temporels :** Là encore, de nombreux types existent :

| Domaine   | Description                                   |
|-----------|---|
| DATE      | date au format 'AAAA-MM-JJ'                   |
| TIME      | heure au format 'hh:mm:ss'                    |
| TIMESTAMP | date et heure au format 'AAAA-MM-JJ hh:mm:ss' |

## 4 Spécification des contraintes d'intégrité

**Clé primaire :** À la création d'une relation, on spécifie qu'un attribut en est une clé primaire en précisant à la suite du domaine le mot clé `PRIMARY KEY`. Par exemple :

```
CREATE TABLE R(a INT PRIMARY KEY, b INT) ;
```

**Clé primaire composite :** Il est aussi possible de déclarer une clé primaire composite en spécifiant après tous les attributs `PRIMARY KEY (cle1, cle2, ...)`. Par exemple :

```
CREATE TABLE R(a INT, b INT, PRIMARY KEY (a,b)) ;
```

**Clé étrangère :** À la création d'une relation, on spécifie qu'un attribut en est une clé étrangère en précisant à la suite du domaine le mot clé `REFERENCES` suivi de la relation et de l'attribut référencés sous la forme `Relation(att)`. Par exemple :

```
CREATE TABLE R1(a INT PRIMARY KEY, b INT) ;  
CREATE TABLE R2(a INT REFERENCES R1(a), c INT) ;
```

**Non-nullité :** En SQL, les attributs n'étant ni une clé primaire, ni référencés par une clé secondaire peuvent être non renseignés. Cela correspond à une case vide dans un tableau. Il existe alors une valeur particulière affectée à ces attributs : `NULL`. Cependant, il est possible de spécifier à la création de table qu'un attribut ne peut pas prendre cette valeur. Pour ce faire, on rajoute le mot clé `NOT NULL` à la suite du domaine. Par exemple :

```
CREATE TABLE R(a INT PRIMARY KEY, b INT NOT NULL) ;
```

**Remarque 1 :** Une clé primaire est toujours non-nulle, pas la peine de le rajouter dans ce cas.

**Remarque 2 :** C'est une bonne habitude à prendre que de déclarer tous les attributs non-nuls.

**Contraintes utilisateur :** Pour spécifier les contraintes utilisateur, on utilise à la création de table, après la déclaration des attributs le mot clé `CHECK` suivi d'une formule booléenne (on verra en détail la syntaxe de ses expressions dans un futur chapitre). Par exemple :

```
CREATE TABLE R(a INT PRIMARY KEY, b INT NOT NULL, c INT NOT NULL, CHECK a<=b AND b<>c) ;
```

## 5 Suppression de tables

On peut supprimer une table d'une BD avec la commande `DROP TABLE Nom_table`. Cependant, si une table sert de référence à une entité présente dans une autre table, cela ne sera pas possible. Il faut donc prendre garde à supprimer les tables **dans le bon ordre**. Par exemple pour supprimer les tables créées dans la deuxième partie on écrira :

```
1 DROP TABLE Emprunt ;  
2 DROP TABLE Livre ;  
3 DROP TABLE Usager ;
```

## 6 Insertion dans une table

Une fois une table créée, on peut y insérer des entités.

**Insertion de ligne :** Pour insérer une nouvelle entité dans une table, on utilise la syntaxe suivante :

```
INSERT INTO Nom_table VALUES (valeur1, valeur2, ...) ;
```

Où `valeur1, valeur2, ...` sont des valeurs respectant les contraintes d'intégrité de chacun des attributs de la table. Par exemple :

```
1 INSERT INTO Usager VALUES ('Dupin', 'Amantine', '5 rue Meslay', '75003', 'Paris', '000000') ;
2 INSERT INTO Usager VALUES ('Kent', 'Clark', '1 rue de la mer', '01630', 'Saint-Genis Pouilly', '000001') ;
3 INSERT INTO Livre VALUES ('Notre dame de Paris', 'Victor Hugo', 'LGF', 2010, '978-2-253-00968-9', 1) ;
4 INSERT INTO Livre VALUES ('Notre dame de Paris', 'Victor Hugo', 'LGF', 2010, '978-2-253-00968-9', 2) ;
5 INSERT INTO Livre VALUES ('Notre dame de Paris', 'Victor Hugo', 'LGF', 2010, '978-2-253-00968-9', 3) ;
6 INSERT INTO Livre VALUES ('Le huitieme sortilege', 'Terry Pratchett', 'Pocket', 1993, '978-2-266-21182-6',
7 4) ;
8 INSERT INTO Livre VALUES ('Le huitieme sortilege', 'Terry Pratchett', 'Pocket', 1993, '978-2-266-21182-6',
9 5) ;
10 INSERT INTO Livre VALUES ('Le huitieme sortilege', 'Terry Pratchett', 'Pocket', 1993, '978-2-266-21182-6',
11 6) ;
12 INSERT INTO Livre VALUES ('Logicomix', 'Apostolos Doxiadis', 'Vuibert', 2018, '978-2-311-10232-1', 7) ;
13 INSERT INTO Livre VALUES ('Logicomix', 'Apostolos Doxiadis', 'Vuibert', 2018, '978-2-311-10232-1', 8) ;
14 INSERT INTO Emprunt VALUES ('000000', 1, '2020-10-05') ;
15 INSERT INTO Emprunt VALUES ('000001', 4, '2020-10-02') ;
16 INSERT INTO Emprunt VALUES ('000001', 7, '2020-10-02') ;
```

Si une entité insérée ne respecte pas les contraintes d'intégrités, une erreur est renvoyée par le SGBD.

**Vérifier ses insertions :** L'affichage des données est l'objet du prochain chapitre sur les bases de données mais dans le but de vérifier que tout se passe bien, on donne la syntaxe basique suivante :

```
SELECT att1, att2, ... FROM Relation
```

où att1, att2, ... sont les attributs de Relation à afficher.

Par exemple :

```
SELECT nom, prenom, cp FROM Usager SELECT titre, isbn FROM Livre
```