Le langage SQL (*Strutured 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 : des **mises à jour** (création de relations, ajout de données, modifications) et des **requêtes** (affichage). Ce chapitre traite de la création de relations et de l'ajout de données et modification de données.

1 La configuration de MySQL dont vous êtes le héros

Ouvrez un terminal et entrez la commande suivante :

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

Si un mot de passe MySQL vous est demandé, c'est que l'installation a déjà été faite et le mot de passe `root` est `0000`.
La chance vous a souri : ***rendez-vous à la partie 1.2.***

Sinon, vous allez devoir combattre pour procéder à une nouvelle installation : ***rendez-vous à la partie 1.1.***

1.1 Nouvelle installation

Assurez-vous que le raspberry est connecté à internet puis entrez les commandes suivantes les unes après les autres dans le terminal (laissez le temps aux commandes de s'exécuter et appelez-moi si un message d'erreur s'affiche).

```
sudo apt update
sudo apt-get update
sudo apt-get install mariadb-server
sudo mysql_secure_installation
```

La dernière commande permet de configurer le logiciel (répondre oui à toutes les question et définir le mot de passe `0000`). Le SGBD MySQL est maintenant installé sur le raspberry et un super utilisateur (appelé `root`) a été configuré.

Vous pouvez maintenant ouvrir MySQL en tant que super utilisateur :

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

Félicitation ! Vous avez terminé l'installation de MySQL et êtes connecté en tant que super utilisateur ! ***Rendez-vous à la partie 1.3.***

1.2 Nettoyage de MySQL

Le fantôme d'un ancien élève a déjà installé MySQL et a déjà (peut être...) travaillé sur votre machine. Nous allons supprimer toute trace de cet odieux personnage.

Les commandes suivantes permettent d'afficher respectivement les utilisateurs enregistrés sur le logiciel et les bases de données existantes :

```
SELECT User FROM mysql.user ;
SHOW DATABASES ;
```

Les commandes suivantes permettent de supprimer respectivement un utilisateur (`Toto`) et une base de données (`ma_base`).

```
DROP USER 'Toto' ;
DROP DATABASE 'ma_base' ;
```

En vous servant des commandes précédentes, supprimez :

- tous les utilisateurs **sauf** `root` ;
- toutes les bases de données sauf `mysql` et `infromation_schema` et `performance_schema`.

Vérifiez que vos actions ont abouti.
Félicitation ! Vous avez vaincu le fantôme de l'ancien élève ! ***Rendez-vous à la partie 1.3****.*

1.3 Je s'appelle `root`

Vous êtes actuellement connecté à MySQl en tant que super utilisateur `root`. Cet utilisateur a la particularité de disposer de tous les droits sur MySQL. Il faut toujours faire très attention lorsqu'on effectue des opérations dans ce cadre car, à la moindre erreur, le fonctionnement du logiciel peut être compromis et la réinstallation est alors notre seul salut.

Afin d'éviter cela, vous allez vous créer un compte utilisateur personnel et utiliser ce compte autant que possible plutôt que `root`. Faites-le en sachant que la commande suivante crée un utilisateur `Toto` de mot de passe `mdp` :

```
CREATE USER 'Toto'@'localhost' IDENTIFIED BY 'mdp' ;
```

Comme on vient de le voir, `root` peut créer de nouveaux utilisateurs. C'est également par lui que nous devrons passer pour créer de nouvelles bases de données et autoriser notre utilisateur personnel à les manipuler.

Exécutez les commandes suivantes en remplaçant `Toto` par votre nom d'utilisateur afin de créer une base de donnée nommée `bibliotheque` et de vous accorder le droit de la manipuler :

```
CREATE DATABASE bibliotheque ;
GRANT ALL PRIVILEGES ON bibliotheque.* TO 'Toto'@'localhost' ;
```

Afin de travailler votre base de donnée toute neuve, quittez MySQL puis reconnectez vous avec votre compte personnel et selectionnez `bibliotheque`.

```
quit ;
```

On vient de quitter MySQL et de retourner dans le terminal Linux.

```
mysql -u Toto -p
```

Une fois votre mot de passe renseigné, vous êtes connecté en tant que vous-même.
Il ne reste plus qu'à sélectionner la base de donnée :

```
USE bibliotheque ;
```

Bravo ! Vous êtes arrivé au bout de cette incroyable aventure. ***Après avoir lu les remarques de la partie 1.4****, vous allez pouvoir commencer le chapitre*

1.4 Quelques remarques pour finir de commencer

Tout ce qui précède n'est pas au programme de terminale NSI. Il est cependant important pour cette année de savoir administrer vos bases de données et votre compte utilisateur correctement sur votre machine.

Terminons cette partie par quelques remarques :

— On sait qu'on se trouve dans le terminal Linux grâce au `pi@raspberrypi:~$` qui s'affiche en début de ligne.

— On sait qu'on est connecté à `MySQL` grâce au `MariaDB [...]>` qui s'affiche en début de ligne.

— Chaque commande SQL se termine par `;` (après un an de Python, attention de ne pas l'oublier !).

— À chaque fois que vous aurez besoin de créer une nouvelle base de donnée ou un nouvel utilisateur, il faudra vous connecter en tant que `root`. Sinon, utilisez votre utilisateur personnel mais rappelez-vous qu'il doit disposer des droits sur la base de donnée que vous voulez utiliser.

— Chaque utilisateur se connecte au SGBD à partir d'une certaine machine toujours précisée. Pour nous, cette machine sera toujours l'ordinateur à partir duquel on travaille ici désigné par `localhost`.

— Pour éviter de trop vous embêter à installer et configurer un serveur SQL, vous pouvez vous entraîner à la maison sur un éditeur en ligne, par exemple `https://sqliteonline.com/`.

Toute 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 donnée ;

— d'une interface SQL en ligne ou locale ;

— à partir d'un fichier lancé avec la commande :

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

2 Création et suppression de tables

Vous êtes normalement connecté à MySQL en tant qu'utilisateur disposant de tous les droits sur la BD `bibliotheque`.

2.1 Un exemple

Nous allons organiser cette base de donnée selon le schéma décrit dans le chapitre précédent, à savoir :

— Usagers(*code* `Int`, *prénom* `String`, *nom* `String`, *date_ naissance* `Date`, *adresse* `String`, *cp* `String`, *ville* `String`) ;

— Livres(*num* `Int`, *titre* `String`, *auteur* `String`, *isbn* `String`, *année* `Int`) ;

— Emprunts(*code* `Int`, *num* `Int`, *date* `Date`) où *code* et *num* font référence aux *code* de *Usagers* et *num* de *Livres*.

Entrez les commandes suivantes :

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


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


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

On reconnaît assez facilement le schéma dans ces commandes.

Remarque : Entre le modèle relationnel et le langage SQL, certains termes peuvent différer. Ainsi, une relation est appelée en SQL une **table** et un attribut une **colonne**. Cela provient du fait que les relations y sont représentées sous forme de tableaux.

> **Création de table :** Pour créer une table `Table` en SQL, la syntaxe est la suivante :
>
> `CREATE TABLE Table(att1 DOM1, att2 DOM2, ...) ;`
>
> On y précise de plus les contraintes d'intégrité comme nous allons le voir ensuite.

2.2 Spécification des contraintes de domaine

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 PRESISION`	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'`

2.3 Spécification des contraintes d'entité

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)) ;
```

2.4 Spécification des contraintes de référence

Clé étrangère : À la création d'une relation, on spécifie qu'un attribut en est une clé étrangère en après la définition de tous les attributs avec la syntaxe `FOREIGN KEY(att) REFERENCES R(att)` où `att` est la clé et `R` la relation référencée. Par exemple :

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

2.5 Spécification des contraintes utilisateur

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. Par exemple :

```
CREATE TABLE R(a INT PRIMARY KEY, b INT, c INT, CHECK a<=b AND b<>c) ;
```

TESTS ET OPÉRATEURS BOOLÉENS : Les tests utilisables sont :

- égalité : `a = b` ;
- inégalités : `a < b` ou `a <= b` etc. ;
- non-égalité : `a <> b` ;
- encadrement : `a BETWEEN c and d` ;
- modèle de texte : `a LIKE modele`
  où `modele` est un modèle de texte pouvant contenir les caractère `'_'` et `'%'` interprétés respectivement comme « un caractère quelconque » et « une suite quelconque de caractère ». Exemples :
  - `s LIKE "Bonjour les ____ !"` teste si le texte `s` est exactement de la forme renseignée où chaque `'_'` peut être remplacé par n'importe quel caractère.
  - `s LIKE "A%"` teste si `s` commence par le caractère `'A'` ;

— `s LIKE "%BAC%"` teste si `s` contient la séquence de caractères `"BAC"` ;

On peut de plus combiner ces tests avec les opérateurs booléens `AND`, `OR` et `NOT`.

Non-nullité : En SQL, les attributs n'étant pas 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 NOT NULL, b INT NOT NULL) ;
```

2.6 Suppression de tables

On peut supprimer une table d'une BD avec la commande `DROP TABLE Nom_table`. Attention 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 :

```
DROP TABLE Emprunt ;
DROP TABLE Livre ;
DROP TABLE Usager ;
```

Créez puis supprimez une table `Inutile` dans votre BD.

3 Ajout, modification et suppression de données dans une table

3.1 Ajout de données

Une fois une table créée, on peut y insérer des entités (ou lignes).

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

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

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

Il est possible d'insérer des lignes une par une mais quand le nombre de données est important, il est plus pratique d'écrire un **fichier SQL**.

Écrire un fichier `initialisation.sql` puis l'exécuter à l'aide de la syntaxe vue à la fin de la partie 1. Attention : la commande donnée est une commande Bash et non SQL, il faut donc avoir quitté MySQL pour que la commande fonctionne.

```
/*
Tenant sur plusieurs lignes, je suis un commentaire
Sans moi bien difficile, d'eclaircir vos mysteres
*/

-- Suppression de toutes les anciennes donnees (commentaire monoligne, en prose)
DELETE FROM Usager ;
DELETE FROM Livre ;
DELETE FROM Emprunt ;

-- Insertion des usagers
INSERT INTO Usager VALUES ('Dupin', 'Amantine', '5 rue Meslay', '75003', 'Paris', '000000') ;
INSERT INTO Usager VALUES ('Kent', 'Clark', '1 rue de la mer', '01630', 'Saint-Genis Pouilly', '000001') ;

-- Insertion dans iivres
INSERT INTO Livre VALUES ('Notre dame de Paris', 'Victor Hugo', 'LGF', 2010, '978-2-253-00968-9', 1) ;
INSERT INTO Livre VALUES ('Notre dame de Paris', 'Victor Hugo', 'LGF', 2010, '978-2-253-00968-9', 2) ;
INSERT INTO Livre VALUES ('Notre dame de Paris', 'Victor Hugo', 'LGF', 2010, '978-2-253-00968-9', 3) ;
INSERT INTO Livre VALUES ('Le 8eme sortilege', 'Pratchett', 'Pocket', 1993, '978-2-266-21182-6', 4) ;
INSERT INTO Livre VALUES ('Le 8eme sortilege', 'Pratchett', 'Pocket', 1993, '978-2-266-21182-6', 5) ;
INSERT INTO Livre VALUES ('Le 8eme sortilege', 'Pratchett', 'Pocket', 1993, '978-2-266-21182-6', 6) ;
INSERT INTO Livre VALUES ('Logicomix', 'Apostolos Doxiadis', 'Vuibert', 2018, '978-2-311-10232-1', 7) ;
INSERT INTO Livre VALUES ('Logicomix', 'Apostolos Doxiadis', 'Vuibert', 2018, '978-2-311-10232-1', 8) ;

-- Insertion dans emprunts
INSERT INTO Emprunt VALUES ('000000', 1, '2020-10-05') ;
INSERT INTO Emprunt VALUES ('000001', 4, '2020-10-02') ;
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
```

3.2 Suppression de données

Il est possible de supprimer une ligne (entité) d'une table.

Suppression de lignes : Pour supprimer une entité identifiée par une certaine valeur de clé primaire d'une table `Table`, on utilise la syntaxe suivante :

```
DELETE FROM Table WHERE cle_primaire = valeur ;
```

Cette syntaxe peut être modifiée pour supprimer plusieurs lignes d'un coup en remplaçant la condition `cle_primaire = valeur` par une condition plus générique :

```
DELETE FROM Table WHERE condition ;
```

Voire même pour supprimer toutes les données de la tables :

```
DELETE FROM Table ;
```

Clark Kent a rendu les livres qu'il a empruntés. La commande permettant de supprimer les lignes correspondantes dans la relation Emprunt est :

```
DELETE FROM Emprunt WHERE code = '000001' ;
```

Vérifiez que la commande a bien fonctionné avec un `SELECT`.

ATTENTION : Peut-on supprimer Amantine de la relation Usager ? Pourquoi ? Essayez de le faire quand même.

3.3 Modification de données dans une table

Il est possible de supprimer une ligne (entité) d'une table.

Modification de ligne : Pour modifier un attribut `att` d'une entité identifiée par une certaine valeur de clé primaire d'une table `Table`, on utilise la syntaxe suivante :

```
UPDATE Table SET att = nouvelle_valeur WHERE cle_primaire = valeur ;
```

Il est possible de modifier les valeurs de plusieurs attributs en même temps, il suffit de séparer les différentes affectations par des virgules. On peut aussi comme avec `DELETE` modifier les valeurs de plusieurs lignes à la fois en remplaçant la condition `cle_primaire = valeur` par une condition plus générique.

```
UPDATE Table SET att1 = nv1, att2 = nv2 ... WHERE condition ;
```

Clark Kent a perdu ses lunette et souhaite qu'on remplace son prénom et son nom dans notre BD par `"Super"` et `"Man"`. On peut le faire avec la commande :

```
UPDATE Usager SET prenom = 'Super', nom = 'Man' WHERE code = '000000' ;
```

Vérifiez que la commande a bien fonctionné avec un `SELECT`.

ATTENTION : Peut-on modifier le prénom d'Amantine dans la relation Usager ? Peut-on modifier son code ? Essayez.