Activité 1 - Installation de MariaDB et création de la base de données Bibliothèque

/home/phil/Documents/LMG/STS_A_2025-2027/PYTHON_STS2/D_BDD_introduction/01_bibliotheque

Table des matières

1. Objectifs....2
2. Contexte....2
II. Installation de MariaDB....2
a. Principe....2
b. Mise à jour du système....2
c. Installation de MariaDB....3
2. Vérification du serveur MariaDB....3
3. Première connexion à MariaDB....3
4. Comprendre l'authentification....3
5. Vérification de la connexion....4
6. Découverte des bases existantes - SHOW DATABASES;....4
III. Création de la bdd - CREATE DATABASE bibliotheque;....4
1. Sélection de la base de données - USE bibliotheque;....5
2. Création de la table auteur - CREATE TABLE auteur (...)....5
a. Passage du modèle objet au modèle relationnel....5
b. Analyse de la définition....5
La colonne id - id INT PRIMARY KEY AUTO_INCREMENT....5
Les colonnes prenom et nom - prenom VARCHAR(100) NOT NULL....6
3. Vérification de la table auteur - SHOW TABLES; DESCRIBE auteur;....6
4. Création de la table livre....6
a. Passage du modèle objet au modèle relationnel....6
5. Analyse de la table livre....7
6. La clé étrangère - FOREIGN KEY (auteur_id) REFERENCES auteur(id)....7
7. Relation entre les deux tables....8
8. Vérification du modèle....8
9. Modèle relationnel obtenu....9
IV. Bilan de l'activité....10

1. Objectifs

Cette activité a pour objectif de préparer un environnement de travail permettant de manipuler une base de données relationnelle depuis un programme Python.
À l'issue de cette activité, vous serez capable de :

- installer MariaDB sous Debian 12 ;
- vérifier le fonctionnement du serveur MariaDB ;
- se connecter à MariaDB avec le compte administrateur ;
- créer une base de données ;
- créer des tables relationnelles ;
- définir une clé primaire ;
- utiliser `AUTO_INCREMENT` ;
- définir une clé étrangère ;
- représenter une relation `1:N` entre deux tables ;
- faire le lien entre un modèle objet Python et un modèle relationnel.

2. Contexte

Le projet consiste à développer une application de gestion de bibliothèque. Une première version du programme a été réalisée en Python à l'aide de la programmation orientée objet. Le modèle comporte notamment deux classes :

```
class Auteur:                    class Livre:
    ...                              ...
```

Un livre est associé à un auteur. La relation entre les objets peut être représentée ainsi :

```
Auteur 1 ———————— N Livre
```

Dans le programme Python, cette relation est représentée par l'attribut : `livre.auteur`

L'objectif de cette activité est de commencer à transformer ce modèle objet en **modèle relationnel**, afin que les données puissent être conservées dans une base de données.

II. Installation de MariaDB

a. Principe

MariaDB est un Système de Gestion de Bases de Données Relationnelles (SGBDR).
Sous Debian 12, MariaDB peut être installé à partir des dépôts Debian. Les opérations d'administration du système nécessitent des privilèges administrateur. Il est recommandé de travailler avec le compte utilisateur habituel et d'utiliser `sudo` lorsque cela est nécessaire.
Il est donc préférable de ne pas travailler en permanence avec le compte Linux `root`.

b. Mise à jour du système

Depuis le compte utilisateur habituel : `sudo apt update`
Puis : `sudo apt upgrade`

c. Installation de MariaDB

Installer le serveur et le client MariaDB :

```
sudo apt install mariadb-server mariadb-client
```

Le paquet `mariadb-server` installe le serveur de bases de données.

Le paquet `mariadb-client` fournit notamment les outils permettant de se connecter au serveur.

2. Vérification du serveur MariaDB

Une fois l'installation terminée, vérifier l'état du service :

```
sudo systemctl status mariadb
```

Le service doit être indiqué comme actif :

```
Active: active (running)
```

Le serveur MariaDB fonctionne alors comme un **service du système**, indépendamment du répertoire dans lequel le terminal était positionné lors de l'installation ou de la connexion.

3. Première connexion à MariaDB

Sur Debian, l'installation standard permet au compte administrateur du système d'accéder au compte administrateur MariaDB à l'aide de `sudo`.

La connexion s'effectue avec :

```
sudo mariadb
```

Une connexion réussie affiche notamment :

```
Welcome to the MariaDB monitor.
...
MariaDB [(none)]>
```

Le prompt : `MariaDB [(none)]>` indique qu'aucune base de données particulière n'est actuellement sélectionnée.

4. Comprendre l'authentification

Dans cette configuration, aucun mot de passe MariaDB n'est nécessaire pour la connexion administrative effectuée avec : `sudo mariadb` Le principe est le suivant :

```
Compte Linux
    |
    | sudo
    v
root Linux
    |
    | authentification locale
    v
root@localhost
    |
    v
MariaDB
```

Il convient de distinguer :

- le compte `root` du système Linux (`root Linux`);
- le compte `root` du serveur MariaDB (`root@localhost`).

Le compte `root` MariaDB sera utilisé pour les opérations d'administration.

Pour une application Python, il sera préférable de créer ultérieurement un **compte MariaDB spécifique à l'application**, plutôt que d'utiliser `root@localhost` qui donne tous les droits sur toutes les bases de données du serveur MariaDB.

5. Vérification de la connexion

Une fois connecté à MariaDB, les commandes SQL suivantes permettent de vérifier l'identité utilisée :

```
SELECT USER(), CURRENT_USER();
```

Une connexion administrative locale peut par exemple produire :

```
+----------------+----------------+
| USER()         | CURRENT_USER   |
+----------------+----------------+
| root@localhost | root@localhost |
+----------------+----------------+
```

Les deux valeurs indiquent ici que la connexion utilise le compte `root@localhost`.

6. Découverte des bases existantes - `SHOW DATABASES;`

La commande suivante permet d'afficher les bases de données présentes sur le serveur :

```
SHOW DATABASES;
```

Sur une installation neuve, plusieurs bases système peuvent être présentes :

```
information_schema
mysql
performance_schema
sys
```

Ces bases sont utilisées par MariaDB pour son fonctionnement et son administration.

Elles ne doivent pas être modifiées dans le cadre de cette activité.

III. Création de la bdd - `CREATE DATABASE bibliotheque;`

Le projet nécessite une base de données spécifique. Créer cette base avec :

```
CREATE DATABASE bibliotheque;
```

MariaDB doit répondre :

```
Query OK, 1 row affected
```

Vérifier sa création avec la commande :

```
SHOW DATABASES;
```

La base `bibliotheque` doit maintenant apparaître dans la liste.

1. Sélection de la base de données - `USE bibliotheque;`

La commande suivante permet de sélectionner la base de données utilisée pour les commandes SQL suivantes :

```
USE bibliotheque;
```

MariaDB indique :

```
Database changed
```

Le prompt devient alors :

```
MariaDB [bibliotheque]>
```

La base `bibliotheque` est désormais la base de données courante.

2. Création de la table `auteur` - `CREATE TABLE auteur (...)`

a. Passage du modèle objet au modèle relationnel

Dans le programme Python, un auteur possède notamment :	Le modèle relationnel correspondant est :

`id`
`prenom`
`nom`
	`AUTEUR`
	`----------`
	`id`
	`prenom`
	`nom`

La table peut être créée avec :

```
CREATE TABLE auteur (
    id INT PRIMARY KEY AUTO_INCREMENT,
    prenom VARCHAR(100) NOT NULL,
    nom VARCHAR(100) NOT NULL
);
```

b. Analyse de la définition

La colonne `id` - `id INT PRIMARY KEY AUTO_INCREMENT`

```
id INT PRIMARY KEY AUTO_INCREMENT
```

`INT` indique que l'identifiant est un entier.

`PRIMARY KEY` indique que cette colonne constitue la **clé primaire** de la table.

Une clé primaire permet d'identifier de manière unique chaque ligne de la table.

`AUTO_INCREMENT` demande à MariaDB de générer automatiquement les identifiants.

Dans le programme Python, les identifiants étaient auparavant générés par le programme :

```
_prochain_id = 1
```

et :

```
self.id = Auteur._prochain_id
Auteur._prochain_id += 1
```

Avec la base de données, cette responsabilité est transférée au SGBD.

Les colonnes `prenom` et `nom` - `prenom VARCHAR(100) NOT NULL`

`prenom VARCHAR(100) NOT NULL`
`nom VARCHAR(100) NOT NULL`
`VARCHAR(100)` indique que la colonne contient une chaîne de caractères dont la longueur maximale est de 100 caractères.
`NOT NULL` interdit l'enregistrement d'une valeur `NULL`.
Un auteur doit donc obligatoirement posséder un prénom et un nom dans ce modèle.

3. Vérification de la table `auteur` - `SHOW TABLES; DESCRIBE auteur;`

Afficher les tables de la base :
`SHOW TABLES;`
La table suivante doit apparaître :
`auteur`
Pour examiner sa structure :
`DESCRIBE auteur;`
Cette commande permet notamment de vérifier :

- les noms des colonnes ;
- les types de données ;
- la présence de la clé primaire ;
- la présence de `AUTO_INCREMENT` ;
- les contraintes `NOT NULL`.

4. Création de la table `livre`

a. Passage du modèle objet au modèle relationnel

Dans le programme Python, un livre possède :	Dans la base de données :

`id`
`titre`
`auteur`
`disponible`
	`LIVRE`
	`----------`
	`id`
	`titre`
	`auteur_id`
	`disponible`

En programmation orientée objet, `auteur` est une référence vers un objet `Auteur`.
Dans une base de données relationnelle, un objet `Auteur` ne peut pas être stocké directement dans une colonne.
La table `livre` va donc stocker **l'identifiant de l'auteur** : c'est à dire un identifiant qui permettra de retrouver l'auteur dans la table AUTEUR

La table peut être créée avec :

```
CREATE TABLE livre (
    id INT PRIMARY KEY AUTO_INCREMENT,
    titre VARCHAR(200) NOT NULL,
    auteur_id INT NOT NULL,
    disponible BOOLEAN NOT NULL DEFAULT TRUE,
    FOREIGN KEY (auteur_id)
        REFERENCES auteur(id)
);
```

5. Analyse de la table `livre`

- La colonne : `id INT PRIMARY KEY AUTO_INCREMENT` joue le même rôle que dans la table `auteur`. Chaque livre possède donc un identifiant unique généré automatiquement par MariaDB.
- La colonne : `titre VARCHAR(200) NOT NULL` contient le titre du livre.
- La colonne : `disponible BOOLEAN NOT NULL DEFAULT TRUE` indique si le livre est disponible. La valeur par défaut : `DEFAULT TRUE` signifie qu'un nouveau livre est considéré comme disponible lorsqu'il est créé, sauf indication contraire.

6. La clé étrangère - `FOREIGN KEY (auteur_id) REFERENCES auteur(id)`

La partie suivante est fondamentale :

```
FOREIGN KEY (auteur_id)
    REFERENCES auteur(id)
```

Elle indique que `livre.auteur_id` est une **clé étrangère** faisant référence à `auteur.id`.

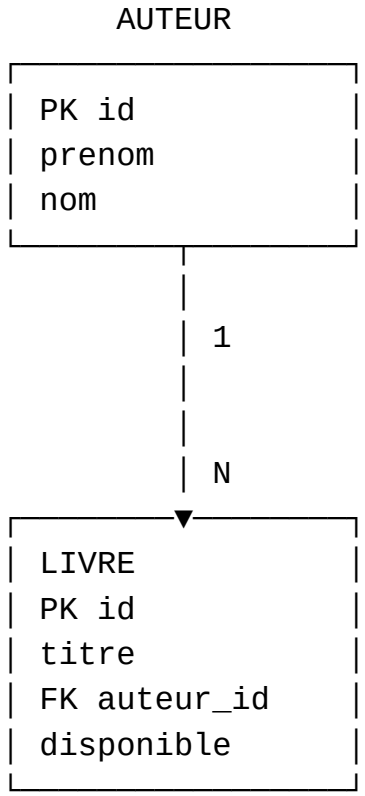
On obtient donc la relation :

```
auteur.id
    ▲
    |
    | référence
    |
livre.auteur_id
```

La clé étrangère permet notamment au SGBD de vérifier qu'un livre fait référence à un auteur existant.

7. Relation entre les deux tables

Le modèle relationnel obtenu est :



La relation est donc : `Auteur 1 ———— N Livre` Un auteur peut être associé à plusieurs livres. Un livre est associé à un seul auteur dans le modèle actuel.

8. Vérification du modèle

Afficher les tables :

```
SHOW TABLES;
```

Les deux tables doivent apparaître :

```
auteur
livre
```

Examiner la structure de `auteur` :

```
DESCRIBE auteur;
```

Puis celle de `livre` :

```
DESCRIBE livre;
```

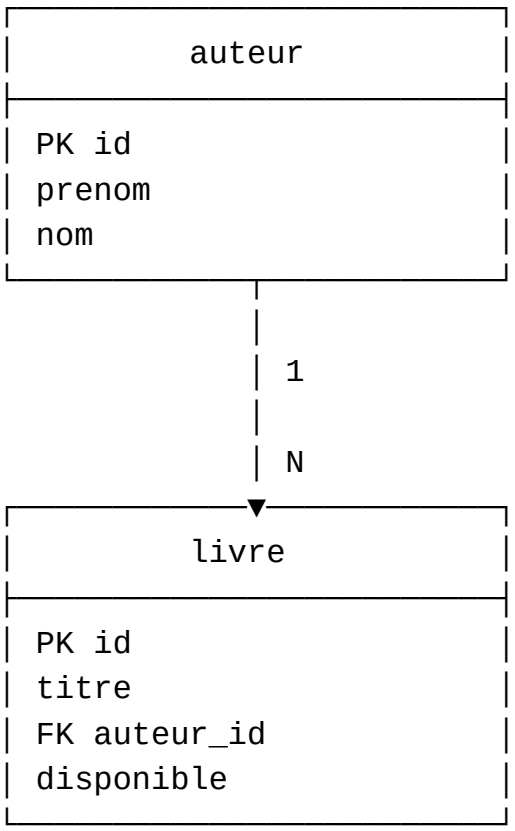
Il est également possible d'afficher la définition complète de la table `livre` :

```
SHOW CREATE TABLE livre;
```

Cette commande permet notamment de vérifier la présence de la contrainte de clé étrangère.

9. Modèle relationnel obtenu

À ce stade, la base de données est organisée de la manière suivante :



La correspondance avec le modèle Python peut être résumée ainsi :

Python	Base relationnelle
`class Auteur`	table `auteur`
`class Livre`	table `livre`
`Auteur.id`	`auteur.id`
`Livre.id`	`livre.id`
`Livre.titre`	`livre.titre`
`Livre.auteur`	`livre.auteur_id`
`Livre.disponible`	`livre.disponible`
référence vers un objet `Auteur`	clé étrangère vers `auteur.id`

IV. Bilan de l'activité

La base de données `bibliotheque` est maintenant créée et possède deux tables :

```
bibliotheque
│
├── auteur
│      ├── id
│      ├── prenom
│      └── nom
│
└── livre
       ├── id
       ├── titre
       ├── auteur_id
       └── disponible
```

Le modèle relationnel reprend donc les principales informations du programme Python tout en utilisant les mécanismes propres à un SGBDR :

- **clé primaire** pour identifier une ligne ;
- **clé étrangère** pour représenter une relation ;
- **`AUTO_INCREMENT`** pour générer les identifiants ;
- **contraintes `NOT NULL`** pour garantir la présence de certaines données ;
- **relation `1:N`** entre les auteurs et les livres.

La prochaine étape consistera à **insérer des données** dans les tables `auteur` et `livre`, puis à utiliser les requêtes `SELECT` et `JOIN` pour exploiter la relation entre les deux tables.