Activité 2 - Manipulation des données de la base Bibliothèque

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

Table des matières

Activité 2 - Manipulation des données de la base Bibliothèque....1
1. Objectifs....2
2. Situation de départ....2
3. Vérification de la base courante....2
4. Insérer des auteurs....3
a. Principe de INSERT INTO....3
b. Ajouter les auteurs du projet....3
5. Vérifier les auteurs....3
6. Sélectionner certaines colonnes....4
Exercice....4
7. Filtrer les résultats avec WHERE....4
a. Exercice....4
8. Insérer les livres....4
Remarque sur L'Étranger....5
9. Vérifier les livres....5
10. Observer la relation entre les tables....5
11. Première utilisation de JOIN....6
12. Comprendre le JOIN....7
13. Comparaison avec Python....7
14. Rechercher un livre particulier....8
15. Rechercher les livres d'un auteur....8
a. Exercice....8
16. Modifier la disponibilité d'un livre....8
17. Rendre le livre disponible....9
18. Attention à la clause WHERE....9
19. Synthèse des principales commandes SQL....9
20. Correspondance avec les méthodes Python....10
21. Travail demandé....10
Partie A — Insertion....10
Partie B — Consultation....10
Partie C — Modification....10
22. État attendu de la base....11

1. Objectifs

Cette activité a pour objectif de manipuler les données de la base `bibliotheque` créée lors de l'activité précédente.
À l'issue de cette activité, vous serez capables de :

- insérer des données dans une table avec `INSERT INTO` ;
- consulter les données avec `SELECT` ;
- sélectionner certaines colonnes ;
- filtrer les résultats avec `WHERE` ;
- utiliser une clé étrangère pour associer deux tables ;
- utiliser `JOIN` pour réunir les informations provenant de plusieurs tables ;
- modifier une donnée avec `UPDATE` ;
- comprendre le lien entre les opérations SQL et les méthodes du programme Python.

2. Situation de départ

La base de données `bibliotheque` contient actuellement deux tables :

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

La relation entre les deux tables est : `Auteur 1 ———————— N Livre`
Les tables ont été créées, mais elles ne contiennent encore aucune donnée.
L'objectif est maintenant de les alimenter.

3. Vérification de la base courante

Ouvrir une connexion à MariaDB : `sudo mariadb`
Sélectionner la base de données : `USE bibliotheque;`
Vérifier les tables : `SHOW TABLES;`
Puis vérifier leur contenu : `SELECT * FROM auteur;`
et : `SELECT * FROM livre;`
Les deux requêtes doivent normalement ne retourner aucune ligne, puisque les tables viennent d'être créées.

4. Insérer des auteurs

a. Principe de `INSERT INTO`

Pour ajouter une ligne dans une table, SQL utilise l'instruction :

```
INSERT INTO nom_table (...)
VALUES (...);
```

Par exemple :

```
INSERT INTO auteur (prenom, nom)
VALUES ('George', 'Orwell');
```

MariaDB doit répondre :

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

b. Ajouter les auteurs du projet

Ajouter les quatre auteurs suivants :

```
INSERT INTO auteur (prenom, nom)
VALUES ('George', 'Orwell');

INSERT INTO auteur (prenom, nom)
VALUES ('Antoine', 'de Saint-Exupéry');

INSERT INTO auteur (prenom, nom)
VALUES ('Albert', 'Camus');

INSERT INTO auteur (prenom, nom)
VALUES ('Jules', 'Verne');
```

Il est important de remarquer que la colonne `id` n'apparaît pas dans les instructions `INSERT`.

MariaDB génère automatiquement les identifiants grâce à : `AUTO_INCREMENT`

5. Vérifier les auteurs

Afficher toutes les colonnes et toutes les lignes :

```
SELECT * FROM auteur;
```

Le résultat devrait être similaire à :

```
+----+---------+------------------+
| id | prenom  | nom              |
+----+---------+------------------+
|  1 | George  | Orwell           |
|  2 | Antoine | de Saint-Exupéry |
|  3 | Albert  | Camus            |
|  4 | Jules   | Verne            |
+----+---------+------------------+
```

Les identifiants sont attribués automatiquement.

Pourquoi est-il préférable de ne pas fournir manuellement les valeurs de `id` lors des insertions ?

6. Sélectionner certaines colonnes

La commande : `SELECT * FROM auteur;` demande toutes les colonnes.

Il est également possible de demander uniquement certaines colonnes :

```
SELECT prenom, nom FROM auteur;
```

Le résultat contient alors uniquement : `prenom  nom`

Exercice

Écrire une requête permettant d'afficher uniquement les identifiants et les noms des auteurs.

7. Filtrer les résultats avec `WHERE`

SQL permet de sélectionner uniquement les lignes répondant à une condition.

Par exemple :

```
SELECT *
FROM auteur
WHERE nom = 'Orwell';
```

Cette requête recherche l'auteur dont le nom est `Orwell`.

Une recherche peut également être effectuée sur l'identifiant :

```
SELECT *
FROM auteur
WHERE id = 3;
```

Cette requête recherche l'auteur dont l'identifiant est `3`.

a. Exercice

Écrire les requêtes permettant de :

1. rechercher l'auteur dont l'identifiant est `2` ;
2. rechercher l'auteur dont le nom est `Camus` ;
3. afficher uniquement le prénom et le nom de `Jules Verne`.

8. Insérer les livres

Les auteurs sont maintenant enregistrés.

Il est possible d'ajouter les livres.

Les quatre livres du projet sont :

```
1984
Le Petit Prince
L'Étranger
Vingt mille lieues sous les mers
```

Les identifiants des auteurs sont actuellement :

```
1 → George Orwell
2 → Antoine de Saint-Exupéry
```

```
3 → Albert Camus
4 → Jules Verne
```

Les livres peuvent donc être ajoutés avec :

```
INSERT INTO livre (titre, auteur_id)
VALUES ('1984', 1);

INSERT INTO livre (titre, auteur_id)
VALUES ('Le Petit Prince', 2);

INSERT INTO livre (titre, auteur_id)
VALUES ('L''Étranger', 3);

INSERT INTO livre (titre, auteur_id)
VALUES ('Vingt mille lieues sous les mers', 4);
```

Remarque sur `L'Étranger`

L'apostrophe dans :

`L'Étranger`

doit être représentée en SQL par deux apostrophes :

`'L''Étranger'`

La première apostrophe permet d'échapper la seconde dans la chaîne de caractères SQL.

9. Vérifier les livres

Exécuter :

```
SELECT * FROM livre;
```

Le résultat devrait être similaire à :

```
+----+------------------------------+-----------+------------+
| id | titre                        | auteur_id | disponible |
+----+------------------------------+-----------+------------+
|  1 | 1984                         |         1 |          1 |
|  2 | Le Petit Prince              |         2 |          1 |
|  3 | L'Étranger                   |         3 |          1 |
|  4 | Vingt mille lieues sous...   |         4 |          1 |
+----+------------------------------+-----------+------------+
```

La valeur `1` de la colonne `disponible` représente `TRUE`.

10. Observer la relation entre les tables

La table `livre` ne contient pas directement le nom de l'auteur.

Elle contient uniquement :

`auteur_id`

Par exemple : `livre`

```
--------------------------------
id | titre           | auteur_id
--------------------------------
1  | 1984            | 1
2  | Le Petit Prince | 2
```

La valeur `1` de `auteur_id` correspond à :

`auteur`

```
-----------------------------
id | prenom | nom
-----------------------------
1  | George | Orwell
```

La base de données établit donc la correspondance :

```
livre.auteur_id
       |
       |
       ▼
auteur.id
```

11. Première utilisation de `JOIN`

Pour obtenir le titre du livre ainsi que le nom de son auteur, il faut réunir les deux tables.

La commande `JOIN` permet cette opération :

```
SELECT
    livre.id,
    livre.titre,
    auteur.prenom,
    auteur.nom
FROM livre
JOIN auteur
    ON livre.auteur_id = auteur.id;
```

Le résultat devrait être similaire à :

```
+----+-----------------------------+---------+------------------+
| id | titre                       | prenom  | nom              |
+----+-----------------------------+---------+------------------+
|  1 | 1984                        | George  | Orwell           |
|  2 | Le Petit Prince             | Antoine | de Saint-Exupéry |
|  3 | L'Étranger                  | Albert  | Camus            |
|  4 | Vingt mille lieues sous...  | Jules   | Verne            |
+----+-----------------------------+---------+------------------+
```

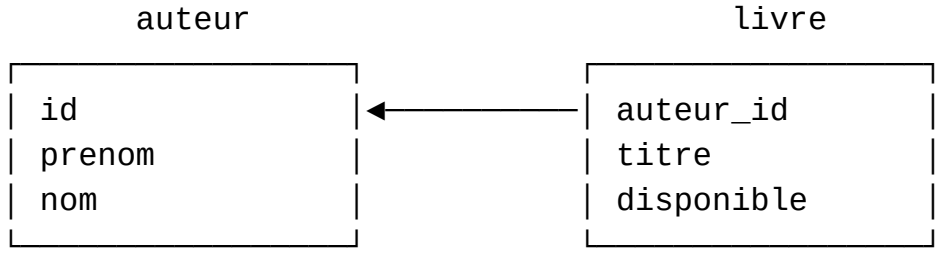
12. Comprendre le `JOIN`

La partie essentielle de la requête est :

```
ON livre.auteur_id = auteur.id
```

Elle indique à MariaDB comment associer les lignes des deux tables.

On peut représenter cette opération ainsi :



La clé primaire :

`auteur.id`

est référencée par la clé étrangère :

`livre.auteur_id`

13. Comparaison avec Python

Dans le programme Python, l'auteur était directement accessible depuis le livre :

`livre.auteur.prenom`

`livre.auteur.nom`

Dans la base de données, le livre ne contient que :

`livre.auteur_id`

Pour obtenir les informations de l'auteur, le SGBD doit effectuer une jointure :

```
JOIN auteur
    ON livre.auteur_id = auteur.id
```

On peut donc établir le parallèle suivant :

Python	SQL
`livre.auteur`	`livre.auteur_id`
`livre.auteur.nom`	`auteur.nom` après `JOIN`
référence vers un objet	clé étrangère
objet `Auteur`	ligne de la table `auteur`

Cette différence constitue un point fondamental du passage de la programmation objet au modèle relationnel.

14. Rechercher un livre particulier

Il est possible de combiner `JOIN` et `WHERE`.

Par exemple :

```
SELECT
    livre.titre,
    auteur.prenom,
    auteur.nom
FROM livre
JOIN auteur
    ON livre.auteur_id = auteur.id
WHERE livre.id = 1;
```

Cette requête permet de retrouver les informations du livre dont l'identifiant est `1`.

15. Rechercher les livres d'un auteur

Il est également possible de rechercher tous les livres d'un auteur.

Par exemple :

```
SELECT
    livre.titre,
    auteur.prenom,
    auteur.nom
FROM livre
JOIN auteur
    ON livre.auteur_id = auteur.id
WHERE auteur.nom = 'Verne';
```

Cette requête recherche tous les livres associés à un auteur dont le nom est `Verne`.

a. Exercice

Écrire une requête permettant d'afficher tous les livres écrits par `Albert Camus`.

16. Modifier la disponibilité d'un livre

Dans le programme Python, l'emprunt d'un livre modifiait :

```
livre.disponible = False
```

En SQL, la modification d'une ligne s'effectue avec `UPDATE`.

Par exemple :

```
UPDATE livre SET disponible = FALSE  WHERE id = 1;
```

Le livre dont l'identifiant est `1` est maintenant considéré comme emprunté. Vérifier :

```
SELECT * FROM livre WHERE id = 1;
```

La colonne `disponible` doit maintenant avoir la valeur `0`.

17. Rendre le livre disponible

Pour simuler la restitution du livre :

```
UPDATE livre
SET disponible = TRUE
WHERE id = 1;
```

Vérifier :

```
SELECT * FROM livre WHERE id = 1;
```

La valeur de `disponible` doit être revenue à `1`.

18. Attention à la clause `WHERE`

La clause `WHERE` est particulièrement importante dans une instruction `UPDATE`.

La requête :

```
UPDATE livre
SET disponible = FALSE;
```

modifierait **tous les livres**.

À l'inverse :

```
UPDATE livre
SET disponible = FALSE
WHERE id = 1;
```

ne modifie que le livre identifié par `1`.

Il est donc indispensable de vérifier les conditions utilisées dans les requêtes `UPDATE` et `DELETE`.

19. Synthèse des principales commandes SQL

Les opérations réalisées dans cette activité peuvent être résumées ainsi :

Besoin	SQL	Exemple de commande complète
Ajouter une ligne	`INSERT INTO`	
Lire des données	`SELECT`	
Filtrer	`WHERE`	
Relier deux tables	`JOIN`	
Modifier une ligne	`UPDATE`	
Créer une base	`CREATE DATABASE`	
Créer une table	`CREATE TABLE`	
Afficher les bases	`SHOW DATABASES`	
Afficher les tables	`SHOW TABLES`	
Examiner une table	`DESCRIBE`	

20. Correspondance avec les méthodes Python

Le programme Python initial comportait notamment les méthodes suivantes :
`ajouter_auteur()` , `trouver_auteur()` , `ajouter_livre()` , `trouver_livre()` , `emprunter_livre()` , `rendre_livre()` , `afficher_auteurs()` , `afficher_livres()`

Les opérations SQL correspondantes commencent à apparaître :

Méthode Python	Opération SQL
`ajouter_auteur()`	`INSERT INTO auteur`
`trouver_auteur()`	`SELECT ... FROM auteur WHERE ...`
`ajouter_livre()`	`INSERT INTO livre`
`trouver_livre()`	`SELECT ... FROM livre WHERE ...`
`emprunter_livre()`	`UPDATE livre`
`rendre_livre()`	`UPDATE livre`
`afficher_auteurs()`	`SELECT ... FROM auteur`
`afficher_livres()`	`SELECT ... FROM livre JOIN auteur`

Cette correspondance sera utilisée dans l'activité suivante pour connecter le programme Python à MariaDB.

21. Travail demandé

Essayer… sans regarder ce qui précède d’effectuer les opérations suivantes dans MariaDB :

Partie A — Insertion

1. Insérer les quatre auteurs.
2. Vérifier le contenu de la table `auteur`.
3. Insérer les quatre livres.
4. Vérifier le contenu de la table `livre`.

Partie B — Consultation

Écrire les requêtes permettant :

1. d'afficher tous les auteurs ;
2. d'afficher tous les livres ;
3. d'afficher un auteur à partir de son identifiant ;
4. d'afficher un livre à partir de son identifiant ;
5. d'afficher tous les livres de Jules Verne ;
6. d'afficher tous les livres avec le prénom et le nom de leur auteur.

Partie C — Modification

1. Emprunter le livre `1984`.
2. Vérifier que sa disponibilité est passée à `FALSE`.
3. Rendre le livre.
4. Vérifier que sa disponibilité est revenue à `TRUE`.

22. État attendu de la base

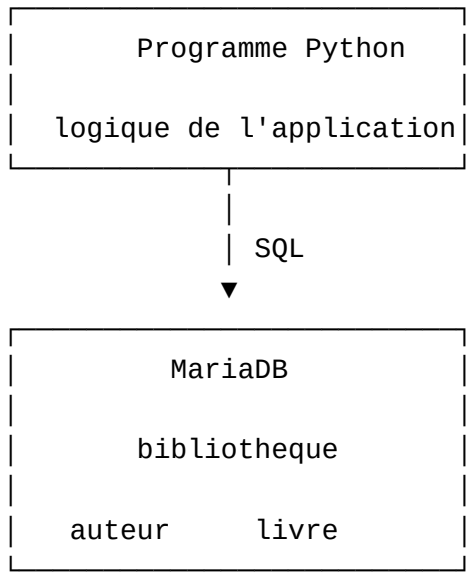
À la fin de l'activité, la base doit contenir :

```
bibliotheque
│
├── auteur
│      ├── 1 – George Orwell
│      ├── 2 – Antoine de Saint-Exupéry
│      ├── 3 – Albert Camus
│      └── 4 – Jules Verne
│
└── livre
       ├── 1 – 1984                          → auteur 1
       ├── 2 – Le Petit Prince                → auteur 2
       ├── 3 – L'Étranger                     → auteur 3
       └── 4 – Vingt mille lieues...          → auteur 4
```

Le modèle relationnel est donc fonctionnel.

La prochaine étape consistera à **connecter Python à MariaDB** afin de remplacer progressivement les listes Python : `self.auteurs: list[Auteur] et self.livres: list[Livre]` par des données stockées de façon persistante dans la base de données.

L'objectif sera alors de faire communiquer les deux couches :



La prochaine activité pourra donc porter sur **l'installation du connecteur MariaDB pour Python, l'établissement d'une connexion et l'exécution des premières requêtes SQL depuis Python**. Ce sera le point de transition entre la manipulation directe de MariaDB et la transformation de votre programme Python existant.