# Bases de données et Python

Bases de données

**Prérequis :** modification des données et module `sqlite3`, jointures, TP SQL.

## Partie 1 – Explorer la base bibliothèque depuis Python

### Rappel : le workflow sqlite3

```python
import sqlite3

# 1. Connexion (ou création) de la base
conn = sqlite3.connect("bibliotheque.db")

# 2. Création d'un curseur
cursor = conn.cursor()

# 3. Exécution d'une requête
cursor.execute("SELECT * FROM Livres")

# 4. Récupération des résultats
resultats = cursor.fetchall()
for ligne in resultats:
    print(ligne)

# 5. Fermeture de la connexion
conn.close()
```

**Exercice 1** *Explorer la base*

Télécharger la base `bibliotheque.db` et la placer dans le même dossier que le script Python.

1. Écrire un programme qui se connecte à la base et affiche tous les livres (titre et auteur) sous la forme :

   ```
   Le Petit Prince - Antoine de Saint-Exupéry
   Les Misérables - Victor Hugo
   ...
   ```

   *Indication :* chaque élément renvoyé par `fetchall()` est un tuple. On accède au titre par `ligne[0]` et à l'auteur par `ligne[1]`.

2. Écrire un programme qui demande à l'utilisateur un genre (avec `input`) et affiche les titres des livres de ce genre.

   **Attention :** ne jamais insérer directement la variable dans la requête avec une f-string. Utiliser un **paramètre** :

```python
genre = input("Genre recherché : ")
cursor.execute("SELECT titre FROM Livres WHERE genre = ?", (genre,))
```

Pourquoi cette précaution est-elle importante ? Rechercher le terme **injection SQL**.

3. Écrire un programme qui affiche le nombre total d'emprunts de chaque élève, sous la forme :

```
Alice Martin : 3 emprunt(s)
Lucas Bernard : 2 emprunt(s)
...
```

*Indication :* utiliser une jointure entre `Emprunts` et `Eleves` avec `COUNT` et `GROUP BY`.

4. Écrire un programme qui affiche la liste des livres actuellement empruntés (non rendus) avec le nom de l'emprunteur.

## Partie 2 – Créer une base de données depuis Python

On souhaite créer de toutes pièces une base de données pour gérer les notes d'un professeur.

### Exercice 2 *Création des tables*

5. Écrire un programme qui crée une base `notes.db` contenant deux tables :

   — **Matieres**(id_matiere INTEGER, nom TEXT NOT NULL) ;

   — **Notes**(id_note INTEGER, #id_matiere INTEGER, eleve TEXT NOT NULL, note REAL NOT NULL, date TEXT NOT NULL, FOREIGN KEY (id_matiere) REFERENCES Matieres(id_matiere)).

   Ne pas oublier `conn.commit()` après les ordres `CREATE TABLE`.

### Exercice 3 *Insertion et requêtes*

6. Ajouter les matières suivantes : Mathématiques (1), NSI (2), Physique-Chimie (3). Puis ajouter les notes suivantes avec `executemany`, qui exécute la même requête pour chaque tuple de la liste :

```python
notes = [
    (1, 2, "Alice Martin", 16.5, "2025-09-15"),
    (2, 2, "Lucas Bernard", 12.0, "2025-09-15"),
    (3, 1, "Alice Martin", 14.0, "2025-09-18"),
    (4, 1, "Lucas Bernard", 9.5, "2025-09-18"),
    (5, 2, "Emma Dubois", 18.0, "2025-09-15"),
    (6, 3, "Emma Dubois", 11.0, "2025-09-20"),
    (7, 2, "Alice Martin", 15.0, "2025-10-10"),
    (8, 1, "Emma Dubois", 13.5, "2025-10-12"),
]
cursor.executemany("INSERT INTO Notes VALUES (?, ?, ?, ?, ?)", notes)
conn.commit()
```

7. Écrire un programme qui affiche, pour chaque matière, la moyenne des notes :

```
Mathématiques : 12.33
NSI : 15.38
Physique-Chimie : 11.00
```

*Indication :* utiliser `AVG`, `JOIN` et `GROUP BY`. Pour formater l'affichage : `f"{moyenne:.2f}"`.

8. Écrire un programme qui affiche la meilleure note et le nom de l'élève correspondant pour chaque matière.

### Exercice 4 *Contraintes d'intégrité*

9. Activer les clés étrangères avec `cursor.execute("PRAGMA foreign_keys = ON")` juste après la connexion. Puis tenter d'insérer une note avec un `id_matiere` inexistant (par exemple 99). Que se passe-t-il ?

   Entourer cette insertion dans un bloc `try … except` pour capturer l'erreur proprement :

```python
try:
    cursor.execute(
        "INSERT INTO Notes VALUES (9, 99, 'Test', 10, '2025-11-01')"
    )
    conn.commit()
except sqlite3.IntegrityError as e:
    print(f"Erreur d'intégrité : {e}")
```

   Les mots `as e` donnent le nom `e` à l'erreur capturée : `print(e)` en affiche le message, qui explique la cause du refus.

10. Écrire une fonction `ajouter_note(conn, id_note, id_matiere, eleve, note, date)` qui insère une note dans la base en gérant les erreurs d'intégrité. La fonction renvoie `True` si l'insertion a réussi, `False` sinon.

## Partie 3 – Pour aller plus loin

**Exercice 5** *Programme interactif*

11. Écrire un programme interactif qui propose un menu à l'utilisateur :

```
=== Gestion de la bibliothèque ===
1. Rechercher un livre par titre
2. Afficher les emprunts en cours
3. Enregistrer un nouvel emprunt
4. Enregistrer un retour
5. Quitter
```

    Le programme doit boucler jusqu'à ce que l'utilisateur choisisse de quitter. Utiliser des fonctions pour structurer le code.

12. Améliorer le programme pour qu'il vérifie, avant d'enregistrer un emprunt, que le livre n'est pas déjà emprunté (c'est-à-dire qu'il n'existe pas d'emprunt sans date de retour pour ce livre).