

Exercice noté (2h) — NSI Terminale — SQL avec Python (SQLite)

Contexte : “Médiathèque du lycée”

Le lycée veut un mini-système pour gérer :

- des **livres**
- des **adhérents**
- des **emprunts**

Vous devez écrire un **script Python** qui :

1. crée une base SQLite,
2. crée les tables,
3. insère les données,
4. exécute des requêtes demandées,
5. affiche des résultats lisibles dans la console.

Outils imposés

- Python (`sqlite3`)
- SQL avec **obligatoirement** : `CREATE TABLE`, `INSERT INTO`, `UPDATE`, `DELETE`, `SELECT`, `INNER JOIN`.

Règles

- 3 tables exactement : `Livre`, `Adherent`, `Emprunt`
 - au moins **10 livres**, **5 adhérents**, **5 emprunts**
 - le script doit être **reproductible** (en relançant, on obtient la même base et les mêmes résultats)
-

Partie A — Modèle relationnel (CREATE TABLE)

Créer les 3 tables suivantes (noms imposés) :

Table `Livre`

Champs recommandés (vous pouvez ajuster légèrement, mais gardez l'idée) :

- `id_livre` (clé primaire)
- `titre` (texte)

- **auteur** (texte)
- **annee** (entier)
- **genre** (texte)
- **disponible** (entier 0/1, par défaut 1)

Table Adherent

- **id_adherent** (clé primaire)
- **nom** (texte)
- **prenom** (texte)
- **classe** (texte)
- **email** (texte, idéalement UNIQUE)

Table Emprunt

- **id_emprunt** (clé primaire)
 - **id_livre** (clé étrangère vers Livre)
 - **id_adherent** (clé étrangère vers Adherent)
 - **date_emprunt** (texte au format YYYY-MM-DD)
 - **date_retour_prevue** (texte YYYY-MM-DD)
 - **date_retour_effective** (texte YYYY-MM-DD ou NULL si pas rendu)
-

Partie B — Insertion des données (INSERT INTO)

Vous devez insérer **exactement** les données suivantes (pour faciliter la correction).

B1 — Livres (10 lignes)

1. (1) 1984 — George Orwell — 1949 — Roman — disponible 1
2. (2) Dune — Frank Herbert — 1965 — SF — 1
3. (3) Le Petit Prince — Antoine de Saint-Exupéry — 1943 — Conte — 1
4. (4) Fondation — Isaac Asimov — 1951 — SF — 1
5. (5) Les Misérables — Victor Hugo — 1862 — Roman — 1
6. (6) L'Étranger — Albert Camus — 1942 — Roman — 1
7. (7) Sapiens — Yuval Noah Harari — 2011 — Essai — 1

8. (8) Harry Potter 1 — J.K. Rowling — 1997 — Fantasy — 1
9. (9) La Horde du Contrevent — Alain Damasio — 2004 — SF — 1
- 10.(10) Vingt mille lieues sous les mers — Jules Verne — 1870 — Aventure — 1

B2 — Adhérents (5 lignes)

1. (1) MARTIN — Lina — TG1 — lina.martin@lycee.fr
2. (2) DUPONT — Sami — TG2 — sami.dupont@lycee.fr
3. (3) NGUYEN — Chloé — TG1 — chloe.nguyen@lycee.fr
4. (4) MOREAU — Adam — TG3 — adam.moreau@lycee.fr
5. (5) BERNARD — Inès — TG2 — ines.bernard@lycee.fr

B3 — Emprunts (5 lignes)

(on est le 2026-02-02 ; certains emprunts doivent apparaître en retard)

1. (1) livre 2 emprunté par adhérent 1 — 2026-01-10 — prévu 2026-01-24 — rendu **NULL**
2. (2) livre 3 par adhérent 2 — 2026-01-12 — prévu 2026-01-26 — rendu 2026-01-20
3. (3) livre 9 par adhérent 3 — 2026-01-18 — prévu 2026-02-01 — rendu **NULL**
4. (4) livre 1 par adhérent 5 — 2026-01-05 — prévu 2026-01-19 — rendu 2026-01-18
5. (5) livre 7 par adhérent 4 — 2026-01-22 — prévu 2026-02-05 — rendu **NULL**

 Logique attendue : si un livre a un emprunt non rendu (`date_retour_effective IS NULL`), alors `Livre.disponible` doit valoir 0.

Partie C — Requêtes (SELECT et INNER JOIN)

Votre script Python doit exécuter et afficher :

C1 — SELECT simple (livres SF)

Afficher `id_livre, titre, annee` de tous les livres dont `genre = 'SF'`, triés par `annee` croissante.

C2 — SELECT avec calcul (nombre de livres dispo)

Afficher le **nombre de livres disponibles** (disponible = 1).

C3 — INNER JOIN (détails des emprunts en cours)

Afficher la liste des emprunts **non rendus** avec :

- `titre` du livre
- nom + prenom de l'adhérent
- `date_emprunt`, `date_retour_prevue`

C4 — INNER JOIN + filtre “en retard”

Afficher les emprunts **en retard** (non rendus ET `date_retour_prevue < '2026-02-02'`) avec les mêmes infos que C3.

Partie D — Modifications (UPDATE et DELETE)

D1 — UPDATE (retour d'un livre)

Consigne : l'emprunt **id_emprunt = 3** vient d'être rendu le **2026-02-02**.

1. Mettre à jour `Emprunt.date_retour_effective`.
2. Mettre à jour `Livre.disponible` du livre concerné à 1.

Ensuite, réafficher la liste des emprunts non rendus (C3) pour vérifier.

D2 — DELETE (suppression contrôlée)

Supprimer l'emprunt **id_emprunt = 2** (il a été saisi en double).

Puis réafficher le nombre total d'emprunts restants.

Livrables attendus

- Un fichier Python : `nsi_sql_mediatheque.py`
 - Le script doit :
 - créer/écraser la base (ex. `mediatheque.db`) ou utiliser `:memory:`
 - afficher clairement les résultats de C1 à D2
-

Barème (/20)

1. CREATE TABLE correct (PK/FK, types, cohérence) — 6 pts
2. INSERT (quantités + données exactes + cohérence FK) — 4 pts
3. Python sqlite3 propre (connexion, curseur, commit, affichages lisibles) — 3 pts
4. C1 (SELECT SF trié) — 2 pts

5. **C2 (compte des dispos)** — 1 pt
 6. **C3 (INNER JOIN emprunts non rendus)** — 2 pts
 7. **C4 (JOIN + retard)** — 2 pts
 8. **D1 (UPDATE emprunt + update livre + vérification)** — 2 pts
 9. **D2 (DELETE + vérification)** — 1 pt
-

Aide minimale (squelette Python possible)

Vous pouvez leur donner (ou non) ce squelette, à compléter :

```
import sqlite3

def afficher(cur, titre, req, params=()):
    print("\n" + "="*10, titre, "="*10)
    cur.execute(req, params)
    for row in cur.fetchall():
        print(row)

con = sqlite3.connect("mediatheque.db")
cur = con.cursor()

cur.execute("PRAGMA foreign_keys = ON;")

# 1) DROP TABLE (optionnel mais utile si relance)
# 2) CREATE TABLE ...
# 3) INSERT INTO ...
# 4) SELECT / JOIN ...
# 5) UPDATE ...
# 6) DELETE ...

con.commit()
con.close()
```