

Modification des données et définition des tables

Fiche de synthèse

① MODIFIER LE CONTENU

```
INSERT INTO Produits
(id, nom, categorie, prix, stock)
VALUES (9, 'Règle', 'Papeterie', 1.80, 120);
```

```
UPDATE Produits
SET prix = 2.80
WHERE nom = 'Cahier A4';
```

```
DELETE FROM Produits
WHERE stock = 0;
```

- `INSERT INTO ... VALUES` ajoute une ligne; les valeurs suivent l'ordre des attributs nommés.
- `UPDATE ... SET ... WHERE` modifie les lignes qui vérifient la condition.
- `DELETE FROM ... WHERE` supprime les lignes qui vérifient la condition.

⚠ Attention. Sans clause `WHERE`, `UPDATE` modifie toutes les lignes de la table, et `DELETE` les supprime toutes : la table reste, mais vide.

Le SGBD refuse toute opération qui viole une contrainte d'intégrité : un `id` déjà utilisé, une valeur hors du domaine, ou la suppression d'une ligne à laquelle une clé étrangère fait référence.

② DÉFINIR UNE TABLE

```
CREATE TABLE Emprunts (
  id_emprunt INTEGER PRIMARY KEY,
  id_livre INT NOT NULL,
  id_adherent INT NOT NULL,
  date_emprunt TEXT NOT NULL,
  duree INT CHECK (duree > 0),
  FOREIGN KEY (id_livre)
    REFERENCES Livres(id_livre),
  FOREIGN KEY (id_adherent)
    REFERENCES Adherents(id_adherent)
);
```

`CREATE TABLE` définit la structure : le nom, le type et les contraintes de chaque attribut. Les types courants sont `INT` ou `INTEGER`, `REAL`, `TEXT` et `DATE`. `DROP TABLE Emprunts;` supprime la table et toutes ses données, de façon irréversible.

Mot-clé	Contrainte	Effet
NOT NULL	domaine	interdit la valeur vide
CHECK (...)	domaine	restreint les valeurs
UNIQUE	relation	interdit les doublons
PRIMARY KEY	relation	clé primaire : unique et non vide
FOREIGN KEY	référence	la valeur désigne une ligne existante

Remarque 1. SQLite, utilisé en TP, ne vérifie les clés étrangères qu'après `PRAGMA foreign_keys = ON;` et n'interdit la valeur vide dans une clé primaire que si elle est déclarée `INTEGER PRIMARY KEY`.

③ UTILISER SQLITE DEPUIS PYTHON

```
import sqlite3

conn = sqlite3.connect("ma_base.db")
cursor = conn.cursor()
# la table Eleves existe déjà dans la base
cursor.execute("""INSERT INTO Eleves
(id_eleve, nom, prenom, classe)
VALUES (1, 'Dupont', 'Marie', 'TA')""")
conn.commit()

cursor.execute("""SELECT nom, prenom
FROM Eleves WHERE classe = 'TA'""")
for ligne in cursor.fetchall():
    print(ligne) # ('Dupont', 'Marie')
conn.close()
```

- `connect` ouvre la base, et crée le fichier s'il n'existe pas.
- `execute` envoie une requête SQL, écrite dans une chaîne de caractères.
- `commit` enregistre les modifications dans le fichier : sans lui, elles sont perdues à la fermeture.
- `fetchall` renvoie la liste des lignes du résultat, chacune sous forme de tuple; `fetchone` renvoie la ligne suivante.
- `close` ferme la connexion sans rien enregistrer.

📌 À retenir.

- `INSERT`, `UPDATE` et `DELETE` modifient le contenu; sans `WHERE`, toutes les lignes sont touchées.
- `CREATE TABLE` définit la structure et déclare les contraintes d'intégrité; `DROP TABLE` supprime la table.
- En Python, on enregistre les modifications avec `commit` avant de fermer la connexion.