Modification des données et définition des tables

Bases de données

① Insertion de données : INSERT

L'ordre `INSERT INTO` ajoute un nouveau n-uplet (une nouvelle ligne) dans une table.

Syntaxe générale :

```
INSERT INTO nom_table (attribut1, attribut2, ...)
VALUES (valeur1, valeur2, ...);
```

Exemple. Ajouter un nouveau produit dans la table Produits :

```
INSERT INTO Produits (id, nom, categorie, prix, stock)
VALUES (9, 'Règle 30 cm', 'Papeterie', 1.80, 120);
```

Les valeurs doivent respecter les types des attributs et les contraintes d'intégrité.

② Modification de données : UPDATE

L'ordre `UPDATE` modifie les valeurs d'un ou plusieurs attributs pour les lignes qui vérifient une condition.

Syntaxe générale :

```
UPDATE nom_table
SET attribut1 = nouvelle_valeur1, attribut2 = nouvelle_valeur2
WHERE condition;
```

Exemple. Le prix du Cahier A4 passe à 2.80 :

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

Attention. Sans clause `WHERE`, **toutes** les lignes de la table sont modifiées.

③ Suppression de données : DELETE

L'ordre `DELETE` supprime les lignes qui vérifient une condition.

```sql
DELETE FROM nom_table
WHERE condition;
```

Exemple. Supprimer les produits dont le stock est nul :

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

Attention. Sans clause `WHERE`, toutes les lignes sont supprimées (la table reste, mais vide). La suppression peut être refusée si des lignes d'une autre table référencent la ligne à supprimer via une clé étrangère.

④ Création d'une table : CREATE TABLE

L'ordre `CREATE TABLE` définit la structure d'une nouvelle table.

```sql
CREATE TABLE Eleves (
    id_eleve        INTEGER PRIMARY KEY,
    nom             TEXT NOT NULL,
    prenom          TEXT NOT NULL,
    classe          TEXT,
    annee_naissance INT CHECK (annee_naissance >= 2000)
);
```

4.1 Types de données courants en SQLite

Type	Description
`INT` ou `INTEGER`	Entier
`REAL`	Nombre à virgule flottante
`TEXT`	Chaîne de caractères
`DATE`	Date (texte `'AAAA-MM-JJ'` en SQLite)

4.2 Déclarer les contraintes

Les contraintes d'intégrité, présentées dans le chapitre *Bases de données relationnelles*, s'écrivent dans `CREATE TABLE`, à la suite d'un attribut ou en fin de définition :

Mot-clé	Contrainte	Effet
`NOT NULL`	Domaine	Interdit les valeurs vides
`CHECK (condition)`	Domaine	Restreint les valeurs acceptées
`UNIQUE`	Relation	Interdit les doublons sur un attribut qui n'est pas la clé primaire
`PRIMARY KEY`	Relation	Désigne la clé primaire : unicité et non-nullité
`FOREIGN KEY (attribut) REFERENCES Table(attribut)`	Référence	Impose que la valeur désigne une ligne existante de l'autre table

La table Eleves ci-dessus déclare déjà une clé primaire, deux attributs `NOT NULL` et un `CHECK`. La table Emprunts déclare en plus deux clés étrangères :

```sql
CREATE TABLE Emprunts (
    id_emprunt      INTEGER PRIMARY KEY,
    id_livre        INT NOT NULL,
    id_adherent     INT NOT NULL,
    date_emprunt    TEXT NOT NULL,
    date_retour     TEXT,
    FOREIGN KEY (id_livre) REFERENCES Livres(id_livre),
    FOREIGN KEY (id_adherent) REFERENCES Adherents(id_adherent)
);
```

Trois particularités de SQLite, le SGBD utilisé en TP :

— SQLite ne vérifie les clés étrangères qu'après la commande `PRAGMA foreign_keys = ON;`, à exécuter au début de chaque connexion ;

— dans une table ordinaire, SQLite accepte une valeur vide dans une clé primaire, sauf si elle est déclarée `INTEGER PRIMARY KEY` ou `NOT NULL` : c'est pourquoi les exemples écrivent `INTEGER`, et non `INT`, devant `PRIMARY KEY` ;

— SQLite accepte une valeur d'un autre type que celui de l'attribut : la chaîne `'vingt-cinq'` est rangée telle quelle dans un attribut `INT`, sauf si la table est déclarée `STRICT`.

⑤ Suppression d'une table : DROP TABLE

```sql
DROP TABLE nom_table;
```

Cette opération est **irréversible** : la table et toutes ses données sont supprimées.

⑥ Interagir avec Python : le module sqlite3

Python intègre le module `sqlite3` qui permet de manipuler une base de données SQLite sans installer de logiciel supplémentaire.

6.1 Se connecter à une base

```python
import sqlite3

conn = sqlite3.connect("ma_base.db")
cursor = conn.cursor()
```

Si le fichier `ma_base.db` n'existe pas, il est créé automatiquement.

6.2 Exécuter des requêtes

```python
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Eleves (
        id_eleve    INTEGER PRIMARY KEY,
        nom         TEXT NOT NULL,
        prenom      TEXT NOT NULL,
        classe      TEXT
```

```
    )
""")

cursor.execute("""
    INSERT INTO Eleves (id_eleve, nom, prenom, classe)
    VALUES (1, 'Dupont', 'Marie', 'Terminale A')
""")
conn.commit()  # Indispensable !
```

La table doit exister avant d'y insérer des lignes : la première requête la crée, sauf si elle existe déjà (`IF NOT EXISTS`).

L'appel à `conn.commit()` est **indispensable** pour enregistrer les modifications dans le fichier. Sans cet appel, les données sont perdues à la fermeture.

6.3 Récupérer les résultats

```
cursor.execute("SELECT nom, prenom FROM Eleves")
resultats = cursor.fetchall()
for ligne in resultats:
    print(ligne)
```

— `cursor.fetchone()` renvoie un seul tuple (la ligne suivante) ;

— `cursor.fetchall()` renvoie une liste de tous les tuples.

6.4 Fermer la connexion

```
conn.close()
```

Cela libère les ressources. Attention : `close()` n'enregistre pas les modifications ; il faut appeler `conn.commit()` avant de fermer la connexion.

⑦ Synthèse

— `INSERT INTO … VALUES` ajoute de nouvelles lignes dans une table ;

— `UPDATE … SET … WHERE` modifie des valeurs existantes (attention sans `WHERE`) ;

— `DELETE FROM … WHERE` supprime des lignes (attention sans `WHERE`) ;

— `CREATE TABLE` définit la structure d'une table ; `DROP TABLE` la supprime ;

— en Python, le module `sqlite3` permet de se connecter, d'exécuter des requêtes et de récupérer les résultats.