SQL et Python

SQL et Python avec *sqlite3*.

L'utilisateur n'entre pas de requêtes SQL

Nous avons étudié le langage SQL qui permet de gérer une base de données par l'intermédiaire de requêtes : `CREATE TABLE`, `INSERT`, `UPDATE`, `SELECT`, etc.
Nous avons utilisé un logiciel de gestion nous permettant d'écrire directement les requêtes en langage SQL et de visualiser le contenu de la base un peu comme si nous utilisions un tableur.
Mais ceci n'est pas l'utilisation habituelle d'une base de données.

Exemple de situation réelle

Une fenêtre – par exemple une fenêtre web – met à disposition des boutons permettant à un utilisateur de consulter le contenu de la base ou d'en modifier le contenu.
Si l'utilisateur souhaite ajouter un acteur, on lui propose un bouton "Ajouter un acteur" qui l'amène à une page adaptée. Sur cette nouvelle page, l'utilisateur trouve des zones de saisie dont la signification est clairement donnée : nom, prénom, date de naissance, biographie...
Un bouton valider permet de conclure l'ajout.
À aucun moment l'utilisateur n'écrit de requête en langage SQL !
Côté serveur, un programme se charge de créer automatiquement les requêtes SQL correspondant aux actions de l'utilisateur. C'est donc un programme, par exemple en Python, qui pilote le SGBD.

module sqlite3

Le module sqlite3 contient les fonctions nécessaires pour permettre à Python de dialoguer avec une BDD.

```
import sqlite3
# par exemple, la bdd est contenue dans le fichier films.db
# le fichier est créé s'il n'existe pas
c = sqlite3.connect("films.db")
```

Au besoin, on peut travailler en mémoire vive, sans enregistrer dans un fichier : `c = sqlite3.connect(":memory:")`

Activation des clés étrangères

La vérification des contraintes de clés étrangères est un surcroît de travail pour le SGBD. On a donc la possibilité de l'activer ou non. Par défaut, cette vérification est désactivée. Pour l'activer :

```
c.execute("PRAGMA foreign_keys = 1")
```

Exemple de requête

On commence par créer un *curseur*. C'est le nom donné à un objet chargé d'agir dans la base de donnée.

```
cursor = c.cursor()
    cursor.execute("""CREATE TABLE ARTISTE (
          idArtiste INTEGER PRIMARY KEY, nom VARCHAR(20),
          prénom VARCHAR(20), biographie TEXT, naissance DATE);""")
```

Remarquez qu'on est libre d'écrire la requête sur plusieurs lignes pour favoriser la lisibilité.

Cas d'une insertion

L'interface web, par exemple, a transmis des données à insérer. On sait donc que l'on doit insérer :

```
prénom = "Bruce"
nom = "Willis"
biographie = "blabla"
naissance = "1955-03-19"
```

Avec ces données on construit la requête :

```
data = (nom, prénom, biographie, naissance)
cursor.execute("INSERT INTO ARTISTE(prénom, nom, biographie, naissance
    ) VALUES(?, ?, ?, ?);", data)
```

Surtout on ne construit pas le texte de la requête avec un formatage habituel : il faut éviter l'**injection SQL**.

Insertion multiple

On a un tableau `tableau` dont chaque item est un tuple comme celui du cadre précédent. On pourrait mettre en place une boucle `for item in tabeau:` mais il vaut mieux utiliser les fonctions du module `sqlite3` :

```
cursor.executemany("INSERT INTO ARTISTE(prénom, nom, biographie,
    naissance) VALUES(?, ?, ?, ?);", tableau)
```

id de l'insertion

Quand on vient d'insérer un item, on ne donne pas son id car c'est le SGBD qui crée tout seul cet id. Mais il peut-être utile de connaître l'id créé. Pour cela :

```
id_du_dernier_inséré = cursor.lastrowid
```

Cas d'une sélection

```
cursor.execute("SELECT * FROM ARTISTE;")
```

On peut ensuite parcourir les items lus :

```
premier = cursor.fetchone()
suivant = cursor.fetchone()
# quand cursor.fetchone() renvoie None, c'est qu'il n'y en a plus
```

```
# tout lire d'un coup et créer un tableau :
tableau = cursor.fetchall()
```

Fin d'utilisation

Pour appliquer les modifications :

```
cursor.commit()
```

Et pour libérer le fichier BDD :

```
c.close()
```