Systèmes de gestion de bases de données

1 Introduction

Dans une base de données, l'information est stockée dans des fichiers, mais à la différence des fichiers au format CSV, il n'est pas possible de travailler sur ces données avec un simple éditeur de texte. Pour manipuler les données présentes dans une base de données (écrire, lire ou encore modifier), il est nécessaire d'utiliser un type de logiciel appelé «*système de gestion de base de données*» très souvent abrégé en SGBD. Il existe une multitude de SGBD : des gratuites, des payantes, des libres ou bien encore des propriétaires. Les SGBD permettent de grandement simplifier la gestion des bases de données :

— les SGBD permettent de gérer la **lecture, l'écriture ou la modification des informations** contenues dans une base de données

— les SGBD permettent de **gérer les autorisations d'accès** à une base de données. Il est en effet souvent nécessaire de contrôler les accès par exemple en permettant à l'utilisateur A de lire et d'écrire dans la base de données alors que l'utilisateur B aura uniquement la possibilité de lire les informations contenues dans cette même base de données.

— les fichiers des bases de données sont stockés sur des disques durs dans des ordinateurs, ces ordinateurs peuvent subir des pannes. Il est souvent nécessaire que l'accès aux informations contenues dans une base de données soit maintenu, même en cas de panne matérielle. Les bases de données sont donc dupliquées sur plusieurs ordinateurs afin qu'en cas de panne d'un ordinateur A, un ordinateur B contenant une copie de la base de données présente dans A, puisse prendre le relais. Tout cela est très complexe à gérer, en effet toute modification de la base de données présente sur l'ordinateur A doit entrainer la même modification de la base de données présente sur l'ordinateur B. Cette synchronisation entre A et B doit se faire le plus rapidement possible, il est fondamental d'avoir des copies parfaitement identiques en permanence. C'est aussi les SGBD qui assurent la maintenance des différentes copies de la base de données.

— plusieurs personnes peuvent avoir besoin d'accéder aux informations contenues dans une base données en même temps. Cela peut parfois poser problème, notamment si les 2 personnes désirent modifier la même donnée au même moment (on parle d'accès concurrent). Ces problèmes d'**accès concurrent** sont aussi gérés par les SGBD.

Comme nous venons de la voir, les SGBD jouent un rôle fondamental. L'utilisation des SGBD explique en partie la supériorité de l'utilisation des bases de données sur des solutions plus simples à mettre en oeuvre ; mais aussi beaucoup plus limitées comme les fichiers au format CSV.

2 Transactions

Supposons que l'on veut supprimer le livre *Les Aventures de Hucklberry Finn* d'ISBN '978-2081509511' de la base de données du CDI. Il faut, en plus de supprimer le livre de la table `livre`, suprimer les relations correspondantes de la table `ecrire` et, si on a supprimer le dernier livre d'un auteur, supprimer cette auteur de la base.
Ce processus s'exprime par plusieurs ordres SQL :

```
-- suprime les relations dans la table ecrire
DELETE FROM ecrire WHERE isbn = '978-2081509511';
-- supprime les auteurs qui n'apparaissent plus dans la table ecrire
DELETE FROM auteur
    WHERE NOT (a_id IN(SELECT a_id FROM ecrire));
-- supprime le livre de la table livre
DELETE FROM livre WHERE isbn = '978-2081509511';
```

Ces trois odres forment un tout et ne doivent pas être dissocier. En, effet considérons la suitation suivante : un élève à reposé le livre en rayon sans le passer par le documentaliste pour le rendre. Il reste alors dans la table emprunt une référence vers l'ISBN du livre.

```
INSERT INTO emprunt VALUES
    (166903291091,'978-2081509511','2020-02-01');

DELETE FROM ecrire WHERE isbn = '978-2081509511';
DELETE FROM auteur
    WHERE NOT (a_id IN(SELECT a_id FROM ecrire));
DELETE FROM livre WHERE isbn = '978-2081509511';
```

On obtient le message d'erreur suivant :

```
FOREIGN KEY constraint failed:
DELETE FROM livre WHERE isbn = '978-2081509511';
```

Nos données sont dans un état incohérent car les deux premiers ordres `DELETE` sont exécutés, retirant de la base l'auteur du livre et la relation entre le livre et son auteur, alors que le dernier ordre à échoué et le livre est toujours présent dans la base. On souhaite donc que que, si l'un des trois ordres échoue, les trois ordres soient annulés. Cette notion fondamentale des SGBD s'appelle une *transaction*.

```
BEGIN TRANSACTION;
DELETE FROM ecrire WHERE isbn = '978-2081509511';
DELETE FROM auteurs
    WHERE NOT (a_id IN(SELECT a_id FROM ecrire));
DELETE FROM livres WHERE isbn = '978-2081509511';
COMMIT;
```

Pour déclarer qu'une suite d'ordres est une transaction, il suffit de la faire précéder de `BEGIN TRANSACTION` et de la conclure avec `COMMIT` afin de la valider.
Si la transaction échoue, on utilise la commande `ROLLBACK` pour l'annuler.

3 Interaction entre SGBD et un programme

3.1 SELECT depuis Python

Un programme interagissant avec une SGBD effectue généralement les actions suivantes :

1. Connexion au SGBD. C'est lors de cette phase que l'on spécifie où se trouve le SGBD, le nom d'utilisateur et le mot de passe.
2. Envoi d'ordres au SGBD. On crée des ordres SQL.
3. On récupère les données correspondant aux résultats dans des structures de données du langage.
4. On peut ensuite exécuter du code Python sur les données récupérées.

Il faut utiliser un module Python qui nous permet de réaliser ces actions. Nous allons ici utiliser `sqlite3`, pour une autre SGBD il faudra utiliser un autre module. Cependant Python utilise une interface *unifiée* d'accès au base de données, ainsi, même pour des SGBD différentes, les méthodes Python sont toujours les mêmes.

```
import sqlite3 as sgbd

#connection à la base de donnée
cnx = sgbd.connect('bdd_cdi.db')

#envoie d'un ordre
```

```
c = cnx.cursor()
c.execute("SELECT * FROM livres WHERE titre LIKE '%Astérix%'")

#récupération et utilisation des données
for l in c.fetchall():
    print(l[0],l[2])

#fermeture de la connexion
cnx.close()
```

La variable `cnx` est un objet représentant la *connexion* à la SGBD. La création d'un curceur `c` à l'aide de la méthode `cursor()` permet d'interagir avec la SGBD. Les deux principales méthodes sont :

- `execute(s ,p)` permet d'exécuter un ordre SQL `s`. Le paramètre `p` est un tableau de valeurs Python.
- `fetchall()` renvoie tous les résultats du dernier ordre passé, sous la forme d'un tableau de n-uplets de valeurs Python. Chaque n-uplet représente une ligne du résultat de la requête.

3.2 Ordres paramétrés

Une fonctionnalité importante est la possibilité de pouvoir insérer dans des ordres SQL des valeurs venant du monde Python, par exemple des saisies d'utilisateur.

```
import sqlite3 as sgbd

cnx = sgbd.connect('bdd_cdi.db')

#chaîne de caractères saisie par l'utilisateur
texte = input ("Texte à rechercher dans le titre :")
#motif à rechercher dans les titres
motif = '%'+texte+'%'

c = cnx.cursor()
c.execute("SELECT titre FROM livres WHERE titre LIKE ?" , [motif])

for l in c.fetchall():
    print(l)

cnx.close()
```

Les `?` prennent les valeurs du tableau donné en deuxième paramètre dans le même ordre.

3.3 Transactions

Les transactions sont une série d'ordres que l'on exécute à l'intérieur d'un block `try`, la commande `ROLLBACK` est quand elle à l'intérieur du block `expect` de sorte si la transaction échoue la base de données est dans son état entérieur.

```
import sqlite3 as sgbd

cnx = sgbd.connect("cdi.db")

c = cnx.cursor()

try:
```

```
    c.execute("BEGIN TRANSACTION")
    c.execute("INSERT INTO emprunt VALUES
    ('978-2081509511',166903291091,'2020-02-01');")
    c.execute("DELETE FROM ecrire WHERE isbn = '978-2081509511';")
    c.execute("DELETE FROM auteur
            WHERE NOT (a_id IN(SELECT a_id FROM ecrire));")
    c.execute("DELETE FROM livre WHERE isbn = '978-2081509511';")
    c.execute("COMMIT")

except sgbd.Error as er:
    print('SQLite error: %s' % (' '.join(er.args)))
    c.execute("rollback")

cnx.close()
```