Chapitre 20 : Base de données relationnelle

I)Introduction

Les bases de données sont très utilisées dans les secteurs de la banque, des assurances, de la finance et de l'administration.
Une *base de données* est un conteneur constitué de *tables* dans lesquelles sont stockées des données. Lorsque ces tables ont des liens entre elles, on parle de *base de données relationnelle*.

Quelques mots à connaître :

– *table* = tableau constitué de lignes et de colonnes,

– *cellule* = intersection d'une ligne et d'une colonne d'une table,

– *valeur* = contenu d'une cellule,

– *attribut* = nom d'une colonne de la table,

– *champ* = attribut ou ensemble d'attributs, (noté $*$ lorsqu'il est constitué de tous les attributs),

– *enregistrement* = ligne de la table, c'est un objet de type *tuple*,

– *clé primaire* d'une table = attribut (ou groupe d'attributs) permettant d'identifier de manière unique tout enregistrement de la table,

– *clé étrangère* d'une table = à la fois attribut de cette table et clé primaire d'une autre table.

Pour manipuler des tables d'une base de données, on utilise un langage de requêtes, appelé langage SQL (=structured query language).

II)Création d'une table

CREATE TABLE nom_table(attributs et type) ;

Exemple :
CREATE TABLE produits (idpro INTEGER,nom TEXT,prix INTEGER)

idpro	nom	prix

III)Modifications des données d'une table

III.1)Insertion d'enregistrements

INSERT INTO nom_table VALUES(valeurs) ;

Exemple :

INSERT INTO produits VALUES
(1,'lampe',40),(2,'bureau',200),(3,'lampe',30),(4,'chaise',50) ;

idpro	nom	prix
1	lampe	40
2	bureau	200
3	lampe	30
4	chaise	50

III.2)Mise à jour des données

- Pour modifier le contenu d'une colonne :

UPDATE nom_table SET attribut=nouvelle_valeur

Exemple :

UPDATE produits SET prix=prix*1.1 ;

idpro	nom	prix
1	lampe	40
2	bureau	200
3	lampe	30
4	chaise	50

↪

idpro	nom	prix
1	lampe	44
2	bureau	220
3	lampe	33
4	chaise	55

- En appliquant un filtre :

UPDATE nom_table SET attribut=nouvelle_valeur
WHERE condition

Exemple :

UPDATE produits SET prix=prix+10 WHERE nom='lampe' ;

idpro	nom	prix
1	lampe	40
2	bureau	200
3	lampe	30
4	chaise	50

↪

idpro	nom	prix
1	lampe	**50**
2	bureau	200
3	lampe	**40**
4	chaise	50

III.3)Suppression d'enregistrements

DELETE FROM nom_table WHERE attribut=valeur ;

Exemple :
DELETE FROM produits WHERE nom='lampe' ;

idpro	nom	prix
1	lampe	40
2	bureau	200
3	lampe	30
4	chaise	50

↪

idpro	nom	prix
2	bureau	200
4	chaise	50

IV)Selection et extraction de données

• Pour selectionner et extraire les enregistrements d'un champ :

SELECT champ FROM nom_table ;

Exemple :
SELECT nom,prix FROM produits ;

idpro	nom	prix
1	lampe	40
2	bureau	200
3	lampe	30
4	chaise	50

↪

lampe	40
bureau	200
lampe	30
chaise	50

• En appliquant un filtre :

SELECT champ FROM nom_table WHERE condition ;

Exemple :
SELECT nom,prix FROM produits WHERE prix>40 ;

idpro	nom	prix
1	lampe	40
2	bureau	200
3	lampe	30
4	chaise	50

↪

bureau	200
chaise	50

V)Jointure

En faisant le produit cartésien d'une table par une autre, on obtient une table, appelée *jointure*, dont les enregistrements sont formés de ceux de la première table, suivis de ceux de la deuxième table.

• Pour selectionner les enregistrements d'un champ de la jointure :

SELECT champ FROM $\underbrace{\text{table 1 INNER JOIN table 2}}_{\text{jointure}}$;

Exemple :

Table produits

idpro	nom	prix
1	lampe	40
2	bureau	200
3	lampe	30
4	chaise	50

X

Table commandes

idcom	ville	idpro
1	Paris	2
2	Valence	4
3	Lyon	1
4	Grenoble	3
5	Paris	4

SELECT * FROM produits INNER JOIN commandes ;

idpro	nom	prix	idcom	ville	idpro
1	lampe	40	1	Paris	2
1	lampe	40	2	Valence	4
...	...	...	...	...	...
4	chaise	50	4	Grenoble	3
4	chaise	50	5	Paris	4

La jointure est constituée de 6 colonnes et de 4x5=20 enregistrements.

SELECT idpro,nom,ville FROM produits INNER JOIN commandes ;

idpro	nom	ville
1	lampe	Paris
1	lampe	Valence
...	...	...
4	chaise	Grenoble
4	chaise	Paris

La jointure est constituée de 3 colonnes et de 4x5=20 enregistrements.

Remarque
Joindre deux tables n'a d'intérêt pratique que si :
– les deux tables ont un attribut en commun,
– on applique un filtre.

• En appliquant un filtre :

SELECT champ FROM table 1 INNER JOIN table 2 ON condition ;

Exemple :
SELECT nom,prix,idcom FROM produits INNER JOIN commandes
ON produits.idpro=commandes.idpro

nom	prix	idcom
lampe	40	3
bureau	200	1
lampe	30	4
chaise	50	2
chaise	50	5

VI)Commandes non exigibles

VI.1)Fonction COUNT

Elle compte le nombre d'enregistrements d'une table :

SELECT COUNT(*) FROM nom_table ;

VI.2)Fonction MAX

Elle retourne la valeur maximale d'un attribut numérique d'une table :

SELECT MAX(attribut) FROM nom_table ;

VI.3)Fonction MIN

Elle retourne la valeur minimale d'un attribut numérique d'une table :

SELECT MIN(attribut) FROM nom_table ;

VI.4)Fonction SUM

Elle retourne la somme des valeurs d'un attribut numérique d'une table :

```
SELECT SUM(attribut)FROM nom_table;
```

VI.5)Fonction AVG

Elle retourne la moyenne d'un attribut numérique d'une table :

```
SELECT AVG(attribut)FROM nom_table;
```

VI.6)Commande ORDER BY

Elle trie les enregistrements d'un champ par ordre croissant selon une colonne de référence donnée :

```
SELECT champ FROM nom_table ORDER BY colonne;
```

VII)Programmer en SQL (hors programme)

La création d'une base de données et la gestion des tables se fait à l'aide du logiciel *sqlite3*.
Pour y avoir accès, on l'importe en tant que module au sein d'un environnement python par la commande import sqlite3.

VII.1)Création d'une base, connexion ou déconnexion à une base de données déjà existante

On utilise la commande :

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

Si la base de données n'existe pas encore, on lui choisit un nom, par exemple : ma_base. [1] C'est un fichier de type *database*.

On remarquera l'utilisation de la fonction *connect* qui renvoie un objet de type *Connection* auquel on attribue un nom, par exemple conn.
Cet objet permet de se connecter à la base de données.

En fin de programme, on ferme cette connexion avec la commande :

```
conn.close()
```

1. On peut aussi créer la base de données dans la mémoire vive (RAM) de l'ordinateur en utilisant la commande : conn=sqlite3.connect(" :memory :").
Mais, les données seront effacées après la déconnexion

VII.2)Création d'un curseur

Une fois connecté à la base de données, on fait appel un autre objet, nommé curseur.
Le curseur est une mémoire tampon, c'est-à-dire une zone de la mémoire vive où l'on peut entreposer temporairement des données, avant de les transférer définitivement dans la base de données.

Pour créer un curseur, on lui donne un nom, par exemple cur, puis on utilise la commande :

cur=conn.cursor()

Ensuite, on peut demander au curseur d'effectuer une tâche, appellée requête [2] (créer une table, insérer une ligne dans une table, importer une table, etc).
Cette requête s'effectue à l'aide de la commande *execute* :

cur.execute(""" ..... requête ....."""")

La requête est placée de part et d'autre entre deux ou trois guillemets doubles.

VII.3)Structure d'un programme de requêtes sql

```
import sqlite3
conn=sqlite3.connect("ma_base.db")
cur=conn.cursor()
.
.
.
REQUETES ET INSTRUCTIONS
.
.
.
conn.commit()
conn.close()
```

La fonction commit() permet de valider les requêtes précédentes.

IX.4)Affichage des données d'une table ou d'un champ

Après l'exécution de la requête SELECT, le curseur met en mémoire une selection d'enregistrements (table entière, colonne ou champ).
Pour les visualiser, on utilise la commande *fetchall()* [3] :

2. query
3. fetch=extraire

```
cur.execute("""'SELECT ....."""')
print(cur.fetchall())
```

On peut rencontrer aussi les commandes :
fetchone() qui n'affiche que le premier enregistrement,
fectchmany(p) qui affiche les p premiers enregistrements.