

Institut Préparatoire aux Études d'Ingénieurs de Tunis



Leila Ben Othman

Deuxième année de classes préparatoires

SM - SP

Année universitaire: 2018 - 2019

SQL (Structured Query Language) et SQLite

Plan

- ① Introduction
- ② SQL: Langage de manipulation de données (LMD)
- ③ SQL: Langage de définition de données (LDD)
- ④ SQLite

Introduction

- SQL (Structured Query Language) est un langage informatique servant à exploiter des bases de données relationnelles.
- Actuellement, le langage SQL est normalisé et constitue le standard d'accès aux bases de données relationnelles.
- Les fonctionnalités du langage SQL sont regroupées en plusieurs catégories:
 - Langage de définition de données
 - Langage de manipulation de données
 - Langage de contrôle d'accès
 - Langage de contrôle des transactions.

Introduction

- Le langage de définition des données permet de créer et de modifier l'organisation des données dans la base (Create table, Drop table, Alter table)
- Le langage de manipulation des données permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans la base (Select, Insert, Update, Delete)
- Le langage de contrôle d'accès permet d'autoriser ou d'interdire l'accès à certaines données, à certaines personnes.
- Le langage de contrôle des transactions permet de commencer et de terminer des transactions.

NB: On va s'intéresser aux deux premières catégories des fonctionnalités du langage SQL.

SELECT

La commande SELECT permet de:

- sélectionner un ou plusieurs attributs d'une table (projection).
- sélectionner des lignes d'une table répondant à un ou plusieurs critères (sélection).
- fusionner des informations provenant de plusieurs tables (produit cartésien, jointure, union, intersection, différence).

⇒ Le résultat de select est une table.

SELECT

Opération	Syntaxe SQL
Projection	(1) select A_1, A_2 from T_1 (2) select * from T_1 (tous les attributs)
Sélection	select * from T_1 where $A_1 = 'a'$
Sélection et projection	select A_2 from T_1 where $A_1 = 'a'$
Produit cartésien	(1) select * from T_1 join T_2 (2) select * from T_1, T_2
Jointure	(1) select A_3 from T_1 join T_2 on $T_1.A_1 = T_2.A_1$ (2) select A_3 from T_1, T_2 where $T_1.A_1 = T_2.A_1$

NB: On peut ajouter le mot-clé **DISTINCT** pour éliminer les doublons.
 Le caractère étoile (*) permet de retourner tous les attributs de la table.

SELECT

Opération	Syntaxe SQL
Intersection	<pre>select A_1 from T_1 intersect select A_1 from T_2</pre>
Union	<pre>select A_1 from T_1 union select A_1 from T_2</pre>
Différence	<pre>select A_1, A_2 from T_1 except select A_1, A_2 from T_2</pre>

Exemple

Soit le schéma relationnel suivant :

- Cours(IdCours,Jour, Heure)
 - SalleCours(IdSalle,IdCours)
 - Etudiants(IdEtudiant, Nom, Prenom, Adresse)
 - Note(IdCours,IdEtudiant, Note)
-
- Les IdCours de tous les cours enseignés.
⇒ `select distinct IdCours from Cours;`
 - Les IdEtudiant de tous les étudiants.
⇒ `select IdEtudiant from Etudiants;`
 - Toutes informations relatives aux notes des étudiants qui suivent le cours 'Algo'.
⇒ `select * from Notes where IdCours='Algo';`

Exemple

- Le jour, l'heure et la salle de chaque Cours.
⇒ `select * from Cours join SallesCours on Cours.IdCours=SallesCours.IdCours;`

⇒ `select * from Cours,SallesCours where Cours.IdCours=SallesCours.IdCours;`
- Toutes les inscriptions possibles
⇒ `select * from Cours join Etudiants.`

Renommage

Opération	Syntaxe SQL
Renommer un attribut	select A_1 as X_1 from T_1
Renommer une table	select * from T_1 as $TableT_1$

NB: Ce Renommage est momentané, c'est-à-dire qui ne dure que dans le cadre de la requête.

Prédicat

prédicat	Syntaxe	Exemple
de comparaison	= != < > <= >=	$X = 2$
d'intervalle	between	X between 1 and 3
de comparaison de texte	like	X like 'l%' X like '%l' X like '%abc%' X like '_abc_'
de test de valeur manquante	is NULL is not NULL	X is NULL
d'appartenance	in	X in (10,20,30)

NB: Le caractère _ désigne un caractère quelconque. Le caractère % désigne une suite de caractères quelconques.

Un prédicat composé est construit à l'aide des connecteurs **and**, **or** et **not**.

Exemple

- Les informations des étudiants dont le nom est 'Titi' ou 'TaTa'.
⇒ `select * from Etudiants where nom in ('Titi','Tata');`
⇒ `select * from Etudiants where nom='Titi' or nom='Tata';`
- Les étudiants tels que les idEtudiant entre 250 et 300.
⇒ `select * from Etudiants where idEtudiant between 250 and 300;`
⇒ `select * from Etudiants where idEtudiant >=250 and idEtudiant <=300;`
- Les étudiants dont le nom commencent par la lettre T et se terminent par la lettre o.
⇒ `select * from Etudiants where nom like 'T%o';`
- Les étudiants dont l'adresse est manquante
⇒ `select * from Etudiants where Adresse is NULL;`

ORDER BY

- La clause ORDER BY permet de préciser les attributs définissant les critères de tri
- Le tri se fera d'abord selon le premier attribut spécifié, puis selon la deuxième, *etc.*
- L'ordre de tri est précisé par ASC (croissant) ou DESC (décroissant). Par défaut, c'est l'option ASC qui est utilisée.

Exemple

- Les id des étudiants triés par ordre croissant.
⇒ `select IdEtudiant,Nom from Etudiants order by idEtudiant;`

- Les noms des étudiants triés par ordre décroissant.
⇒ `select Nom from Etudiants order by Nom DESC;`

- Les id et les noms des étudiants triés selon le nom par ordre croissant ensuite selon les id par ordre décroissant.

⇒ `select Nom,idEtudiant from Etudiants order by Nom
ASC,idEtudiant DESC;`

⇒ `select Nom,idEtudiant from Etudiants order by Nom,idEtudiant
DESC;`

Fonctions de calcul

- Au-delà des opérateurs relationnels, des fonctions de calculs existent. Les fonctions implantées en SQL sont :

Fonction	Rôle
COUNT	Nombre de valeurs
SUM	Somme des valeurs
MIN	Plus petite valeur
MAX	Plus grande valeur
AVG	Moyenne des valeurs

Exemples

- Le nombre d'étudiants de la base.
⇒ `select count(IdEtudiant) from Etudiants`
- Le nombre de cours.
⇒ `select count(distinct IdCours) from Cours;`
- Le nombre de Notes égales à 'A'.
⇒ `select count(IdEtudiant) from Notes where Note='A';`
⇒ `select count(*) from Notes where Note='A';`
- Le nombre d'étudiants ayant une note égale à 'A'.
⇒ `select count(distinct IdEtudiant) from Notes where Note='A';`
- Le nombre d'étudiants ayant une note égale à 'A' en 'BD'.
⇒ `select count(IdEtudiant) from Notes where Note='A' AND IdCours='BD'`

Agrégat - Fonctions d'agrégation

- Un Agrégat est un regroupement, c'est-à-dire un sous-ensemble des lignes d'une table ayant la même valeur pour un attribut donné.
- Un regroupement est déterminé par la clause GROUP BY suivie du nom du ou des attributs sur lesquels s'effectuent le regroupement.
- A la place des noms des attributs, il est possible de spécifier des fonctions de calcul sur les attributs. Dans ce cas, les fonctions de calcul opèrent sur chaque groupe (d'où le nom fonctions d'agrégation).

Exemples

- Soit la table *employe* représentant les employés d'une entreprise:

IdEmployé	Nom	Prénom	salaire	IdDept
1	X	PrenomX	5000	500
2	Y	PrenomY	5050	501
3	Z	PrenomZ	4000	500
4	W	PrenomW	6050	501
5	A	PrenomA	8000	500

- Le salaire moyen.
- Le salaire moyen par département.

Exemples

- Le salaire moyen.
 $\Rightarrow$ `select AVG(salaire) AS SalaireMoyen`
`from employe`

SalaireMoyen
5620

- Le salaire moyen par département.
 $\Rightarrow$ `SELECT IdDept, AVG(salaire) AS SalaireMoyenDept`
`from employe`
`group by IdDept`

IdDept	SalaireMoyenDept
500	5666.666666666667
501	5550.0

NB: Ici, on a effectué un renommage d'attribut.

Exemples

- L'employé qui a le salaire maximale.

```
select *
from employe
where salaire=
(select max(salaire)
from employe)
```

IdEmployé	Nom	Prénom	salaire	IdDept
5	A	PrenomA	8000	500

Exemples

- Les employés tels que le salaire est supérieur ou égal au salaire moyen des employés de l'entreprise.

⇒ select *

from employe

where salaire >=

(select avg(salaire)

from employe)

IdEmployé	Nom	Prénom	salaire	IdDept
4	W	PrenomW	6050	501
5	A	PrenomA	8000	500

La clause Having

- La clause HAVING permet de spécifier une condition de restriction sur les groupes, c'est-à-dire, elle sert à éliminer certains groupes, comme WHERE sert à éliminer des lignes
- Les départements tels que le salaire moyen de ses employés est supérieur à 5550.
 $\Rightarrow$ select idDept ,avg(salaire) as SalaireMoyenDept
 from employe
 group by idDept
 having avg(salaire)>5550

IdDept	SalaireMoyenDept
500	5666.666666666667

Exemples

- Les départements comportant au moins 3 employés.

```
select idDept from employe
group by idDept
having count(*)>=3
```

IdDept
500

- Les noms des employés des départements comportant au moins 3 employés.

```
select nom from employe
where idDept in
(select idDept from employe group by idDept having count(*)>=3);
```

Nom
Z
A
X

Expréssion mathématique, opérateur de concaténation

- On peut mettre une expression mathématique dans une requête SQL.
 - Les employés avec un salaire multiplié par deux.
`select nom, salaire*2 as salairedeux from employe;`
- On peut également utiliser l'opérateur de concaténation `||`.
 - La liste des prénoms et des noms des employés concaténés et séparés par un espace.
`select Prenom || " " || nom from employe;`

SELECT - Syntaxe complète

SELECT [DISTINCT] liste d'attributs, expressions
FROM liste de tables
[WHERE prédicat simple ou composé]
[GROUP BY attributs de groupe]
[HAVING condition de groupe]
[ORDER BY liste des attributs [ASC | DESC]]

NB: Les clauses mises entre [] sont facultatives.

Exists/Not Exists

- Les conditions EXISTS et NOT EXISTS s'utilisent de la manière suivante:

```
SELECT * FROM Table  
WHERE [NOT] EXISTS (sous-requête)
```

- Une condition avec EXISTS est vraie si la sous-requête renvoie au moins une ligne.
- Une condition avec NOT EXISTS est vraie si la sous-requête ne renvoie aucune ligne.

Exemple

- Soit la table *Département* représentant les départements de l'entreprise auxquels les employés sont rattachés:

IdDept	Libellé
500	Finance
501	Marketing
502	Ressources Humaines

- Les départements comportant au moins un employé.
- Les départements (fictifs) ne comportant aucun employé.

Exemple

- Les départements comportant au moins un employé.

⇒ select idDept

from employe

group by idDept

having count(*)>=1;

⇒ select idDept

from departement as d

where exists

(select * from employe as e

where e.idDept=d.idDept);

IDdept
500
501

NB: Ici, on a effectué un renommage des tables.

Exemple

- Les départements (fictifs) ne comportant aucun employé.

```
⇒ select idDept
from departement as d
where not exists
(select *
from employe as e
where e.idDept=d.idDept);
```

<u>IdDept</u>
502

Exemple

- Les étudiants inscrits à tous les cours (Division)

⇒ On sélectionne l'étudiant tel qu'il n'existe pas un cours pour lequel l'étudiant n'est pas inscrit.

⇒ `select * from Etudiants
where not exists
(select * from Cours
where not exists
(select * from Notes
where Notes.idCours=Cours.idCours and
Etudiants.idEtudiant=Notes.idEtudiant));`

Insert

- La commande INSERT permet d'ajouter de nouvelles lignes à une table
- **Syntaxe:**

```
INSERT  
INTO nom_table [ (liste des attributs) ]  
VALUES (liste de valeurs)
```

Exemples:

- insert into Etudiants(IdEtudiant,Nom,Prenom,Adresse) values (100,'Toto','prenomToto', 'Tunis')
- insert into Etudiants values (200,'Tata','prenomTata', 'La Manouba')

Delete

- La commande DELETE permet de supprimer des lignes d'une table.
- L'absence de clause WHERE entraîne la suppression de toutes les lignes de la table.
- **Syntaxe:**

```
DELETE  
FROM table  
[ WHERE prédicat ]
```

- **Exemples:**
 - Delete from Etudiants
 - Delete from Etudiants where IdEtudiant=100

Update

- La commande UPDATE permet de modifier les valeurs d'attributs de lignes existantes.
- L'absence de clause WHERE entraîne la modification de toutes les lignes de la table selon les affectations mentionnées.
- **Syntaxe:**

```
UPDATE Table  
SET liste d'affectations [ WHERE prédicat ]
```

- **Exemples:**
update Etudiants set Adresse='Sousse'
update Etudiants set Adresse='Sousse' where IdEtudiant=100

Create table (1)

- La commande CREATE TABLE permet de définir le schéma d'une table avec les contraintes d'intégrité.
- Syntaxe:

```
CREATE TABLE table  
(  
attribut type [ NOT NULL [UNIQUE] ] [ DEFAULT valeur ]  
[ PRIMARY KEY ]  
[ FOREIGN KEY REFERENCES table ]  
[ CHECK condition ])
```

- Les contraintes d'intégrité:
 - NOT NULL valeur null impossible
 - UNIQUE unicité des valeurs d'un attribut
 - PRIMARY KEY clé primaire
 - FOREIGN KEY clé étrangère
 - CHECK plage ou liste de valeurs CHECK (Adresse IN ('Tunis', 'Sousse', 'Sfax') ,

Create table (2)

- **NB:** La contrainte PRIMARY KEY implique les deux contraintes NOT NULL et UNIQUE.
- **Exemples:**
 - create table if not exists Etudiants(IdEtudiant integer primary key, Nom text, Prenom text, Adresse text)
 - create table if not exists Cours(IdCours text, Jour text, Heure text, primary key(IdCours, Jour))
 - create table if not exists SallesCours(IdCours text, IdSalle text, primary key (IdCours, IdSalle))
 - create table if not exists Notes(IdCours text, IdEtudiant integer, Note text, primary key(IdCours, IdEtudiant), foreign key (IdCours) references Cours(IdCours), foreign key (IdEtudiant) references Etudiant(IdEtudiant))

Drop/ Alter table

- La commande DROP permet de supprimer une table.
- Syntaxe:

Drop table Notes

- La commande Alter permet de modifier le schéma d'une table. On n'abordera que la modification du nom de la table.
- Syntaxe:

Alter table Etudiants rename to Etud

SQLite

- **L'import de la bibliothèque sqlite de python:**

```
import sqlite3 as sq
```

- **La connexion avec la base (création et ouverture):**

```
ma_base=sq.connect("BASE_Cours_Etud_Notes.sqlite")
```

Cette commande permet d'ouvrir le fichier 'BASE_Cours_Etud_Notes.sqlite.sqlite'. Si le fichier n'existe pas, il sera créé.

NB: *ma_base* représente le nom logique de la base. Tandis que "BASE_Cours_Etud_Notes.sqlite" est le nom physique de la base.

SQLite

- **Création d'un curseur où sera stockée une version temporaire du travail:** curseur est un objet de type cursor qui permet d'effectuer des requêtes SQL depuis python.

```
curseur = ma_base.cursor()
```

- **Validation de toutes les transactions (opérations):**

```
ma_base.commit()
```

- **La fermeture de la base (déconnexion):**

```
curseur.close()  
ma_base.close()
```

SQLite

- **Création des tables (schémas des tables ainsi que les contraintes d'intégrités):**

- `curseur.execute("create table if not exists Etudiants(IdEtudiant integer primary key,Nom text,Prenom text,Adresse text)")`
- `curseur.execute("create table if not exists Cours(IdCours text,Jour text,Heure text, primary key(IdCours,Jour))")`
- `curseur.execute("create table if not exists SallesCours(IdCours text,IdSalle text, primary key (IdCours,IdSalle))")`
- `curseur.execute("create table if not exists Notes(IdCours text,IdEtudiant integer,Note text, primary key(IdCours,IdEtudiant), foreign key (IdCours) references Cours(IdCours), foreign key (IdEtudiant) references Etudiant(IdEtudiant))")`

SQLite

- Insertion enregistrement par enregistrement

- `curseur.execute("insert into Etudiants (IdEtudiant,Nom,Prenom,Adresse) values (100,'Toto','prenomToto', 'Tunis'))"`
- `curseur.execute("insert into Etudiants values (100,'Toto','prenomToto', 'Tunis'))"`
- `curseur.execute("insert into Etudiants values(?,?,?,?)", (100,'Toto','prenomToto', 'Tunis'))`

ici, Adresse sera manquante:

- `curseur.execute("insert into Etudiants (IdEtudiant,Nom,Prenom) values (100,'Toto','prenomToto')")`

SQLite

- Insertion de plusieurs enregistrements en même temps (à partir d'une liste)

```
L=list()
L.append((100,'Toto','prenomToto', 'Tunis'))
L.append((200,'Tata','prenomTata', 'La Manouba'))
L.append((300,'Titi','prenomTiti', 'Kairouan'))
curseur.executemany("insert into Etudiants values (?,?,?,?)",L)
```

SQLite

- **Modification d'enregistrement(s):**

```
curseur.execute("update Etudiants set Adresse='Sousse' where  
idEtudiant=100 ")
```

- **Suppression d'enregistrement(s):**

```
curseur.execute("delete from Etudiants where idEtudiant=200 ")
```

- **Modification du nom d'une table:**

```
curseur.execute("Alter table Etudiants rename to Etud")
```

- **Suppression d'une table:**

```
curseur.execute("drop table Notes")
```

SQLite

- **Interrogation (Recherche) des données:**

```
curseur.execute("select * from Etudiants")
```

- **Récupérer le résultat d'une recherche**

- Manipuler directement le curseur.

```
for i in curseur:
    print(i)
```

NB: i est un tuple et représente une ligne du résultat de la requête.

SQLite

- Manipuler les lignes du curseur ligne par ligne avec `fetchone()`.

```
while True:
    T=curseur.fetchone()
    if T!=None:
        print(T)
    else:
        break
```

NB: De même, la méthode `fetchone()` retourne un tuple qui représente une ligne du résultat de la requête.

- Manipuler le résultat du curseur à travers une liste avec `fetchall()`.

```
L=curseur.fetchall()
for i in L:
    print(i)
```

NB: La méthode `fetchall()` retourne une liste comportant toutes les lignes (sous forme de tuples) résultat de la requête.