

LES BASES DE DONNÉES: PASSAGE À LA PRATIQUE

MODULE : SQLITE3 DE PYTHON

INTRODUCTION(1)

La plupart des SGBD fonctionnent sur le mode classique **client-serveur**. C'est le cas par exemple pour :

- ☐ PostgreSQL (libre) ;
- ☐ MySQL (libre ou propriétaire) → fourni par exemple dans la suite WAMP (Windows, Apache, MySQL, PHP) ou LAMP (Linux...) fournissant un serveur Web PHP + BD clef en main.
- ☐ Oracle (propriétaire)

Pour ces SGBD, il faut :

- ☐ installer la partie « **serveur** » du SGBD
- ☐ démarrer le serveur
- ☐ utiliser le client fourni, ou en trouver un qui nous convient.

INTRODUCTION (2)

On va utiliser **SQLite** qui a des caractéristiques particulières:

- ☐ Pas de serveur, juste un client
- ☐ Chaque base de données est juste un fichier. Pour se connecter, on spécifie juste le nom du fichier

AVANTAGES DE SQLITE

- ❑ Très léger (intégré dans Python, module SQLite)
- ❑ Installation très facile et administration inexistante
- ❑ Chaque base peut être distribuée très facilement (c'est juste un fichier)
- ❑ Du domaine public, c-à-d utilisable partout sans restriction légale (free)

INCONVÉNIENTS DE SQLITE

- ❑ Ne convient pas à la gestion de gros volumes de données
- ❑ Certaines fonctionnalités d'interrogation sont manquantes
- ❑ Pas de gestion des utilisateurs, contrôle d'accès, etc.
- ❑ **En résumé** : suffisant pour de petites bases « embarquées » mais pas réaliste sur une application nécessitant des fonctionnalités avancées et de gros volumes de données.

CRÉATION DE BASE DE DONNÉES DEPUIS PYTHON

6

Atef Boujelben

Pour créer une nouvelle base via Python, il suffit de :

- ❑ Importer le module **sqlite3**
- ❑ Se connecter/créer une base
- ❑ Fermer la connexion après transaction (se déconnecter)

```
import sqlite3
nom_logique= sqlite3.connect('nom_base.db')
##Si le fichier nom_base.db existe il se connecte, sinon il le crée.
##Pour fermer la base, il suffit d'écrire:
nom_logique.close()
```

Exemple:

```
import sqlite3
cnx= sqlite3.connect('baseP.db')
cnx.close()
```

CRÉATION D'UN CURSEUR

- ❑ A travers la connexion, on a accès à un objet de type `cursor` qui permettra d'effectuer des requêtes SQL depuis python.
- ❑ La création d'un curseur se fera à partir de la commande

```
curseur=nom_logique.cursor()
```

L'exécution d'une requête se fera par la commande

```
curseur.execute(""" Requête_SQL """)
```

La validation de la transaction se fera à l'aide de la commande:

```
nom_logique.commit()
```

CRÉATION DE TABLES EN SQL

- Une fois la base est créée, il faut l'alimenter, c'est-à-dire créer des tables et les remplir par des données concrètes pour pouvoir effectuer des requêtes.

```
CREATE TABLE nom_de_table(  
    attribut_1 type_1 [PRIMARY KEY],
```

```
    ...
```

```
    attribut_n type_n
```

```
#contrainte clé primaire
```

```
[CONSTRAINT nom_const PRIMARY KEY(liste_attributs)],
```

```
#contrainte clé étrangère
```

```
[CONSTRAINT nom_const FOREIGN KEY(liste_attributs)  
REFERENCES nom_table(liste_attributs)],
```

```
#contrainte d'intégrité
```

```
[CONSTRAINT nom_const CHECK(condition)] );
```

CRÉATION DE TABLES EN SQL:

LES CONTRAINTES D'INTÉGRITÉ

❑ Une contrainte est une règle qui permet de contrôler la validité ou la cohérence des valeurs. On distingue:

✓ **La contrainte attribut (colonne):** Elle concerne un attribut précis. Elle est déclarée sur la ligne de définition de l'attribut en question. Une contrainte peut avoir un nom dans la table(relation). L'avantage de donner un nom est la possibilité de la manipuler (supprimer , modifier)

✓ **La contrainte table:** on utilise une contrainte table lorsqu'elle concerne plusieurs attributs

❑ Les différentes contraintes sont Primary key, Foreign key, NOT NULL, UNIQUE, CHECK, ..

CRÉATION DE TABLES EN SQL: EXEMPLE

```
create table IF NOT EXISTS Etudiant(idEtud INTEGER  
CONSTRAINT pk PRIMARY KEY autoincrement, Nom  
varchar(20), Prenom varchar(20), Adresse Text, CONSTRAINT  
NOT_Null NOT NULL(Nom, Prenom, Adresse))
```

Cette instruction crée une table intitulée **Etudiant** contenant 4 champs :

- ❑ id_etud: entier (c'est la clé primaire qui va être incrémenté automatiquement)
- ❑ NOM : chaîne d'au plus 20 caractères.
- ❑ PRENOM : chaîne d'au plus 20 caractères.
- ❑ ADRESSE : Text

CRÉATION TABLES SQLITE/PYTHON

11

Atef Boujelben

```
import sqlite3
```

```
cnx=sqlite3.connect('baseX.sq3')
```

```
# on crée un tampon où sera stockée une version temporaire de
```

```
#travail de la base de donnée, que l'on écrira ensuite dans le fichier.
```

```
curseur=cnx.cursor()
```

```
curseur.execute(""" create table IF NOT EXISTS Etudiant(idEtud INTEGER
```

```
PRIMARY KEY autoincrement,Nom varchar(20), Prenom varchar(20),
```

```
Adresse Text) """)
```

```
cnx.commit() # on sauvegarde notre travail dans le fichier de la base
```

```
#cnx.rollback() # on peut annuler les dernières modifications
```

```
cnx.close() # on ferme la connexion
```

LES TYPES DE DONNÉES DANS LES BASES

12

Atef Boujelben

Type	Description
NUMBER(n,[d])	Nombre de n chiffres [d (optionnel) après la virgule]
SMALLINT	Entier signé de 16 bits
INTEGER	Entier signé de 32 bits
FLOAT	Nombre à virgule flottante
DATE	Date format jj/mm/aa
TIME	Heure
CHAR(l)	Chaîne de caractères de longueur fixe / (l<16383)
VARCHAR(n)	Chaîne de caractères de n caractères maximum (n<16383)
Text	Un texte

SUPPRESSION DE TABLE

En SQL (Structured Query Language):

```
DROP TABLE nom_table;
```

Exemple:

```
import sqlite3
nom_logique=sqlite3.connect('baseX.sq3')

curseur= nom_logique.cursor()
curseur.execute(""" DROP TABLE Etudiant """)
nom_logique.commit()
nom_logique.close()
```

INSERTION DE DONNÉES

En SQL:

INSERT INTO nom_table **VALUES**(liste_valeurs):

En Python:

```
curseur.execute(""" INSERT INTO Etudiant( Nom, Prenom)
VALUES (" Arbi " , " mohamed" ) """)
```

Ou encore tous les champs avec:

```
curseur.execute(""" INSERT INTO Etudiant VALUES (2,"ben
salah", "Saleh", "Sfax ") """)
```

INSERTION MULTIPLES

❑ Il est également possible de faire plusieurs insert en une seule fois avec la fonction `executemany`:

```
etud= []  
etud.append(("Selmi","ali",))  
etud.append((" Arbi " , "mariem"))  
etud.append((" Arbi " , "Arij" ))  
curseur.executemany("""INSERT INTO Etudiant(nom, prenom)  
VALUES(?,?) """, etud)
```

MIS À JOURS DE DONNÉES

Pour modifier les tuples:

```
curseur.execute("""UPDATE Etudiant  
    SET ADRESSE ="Tunis", nom="Saleh"  
    WHERE id = 2""")
```

LECTURE DE DONNÉES

- ✓ Récupération d'une ligne unique (tuple) à l'aide de la méthode `fetchone`:
 - ✓ `curseur.execute("""SELECT nom, prenom FROM Etudiant""")`
`e1 = curseur.fetchone()`
`print(e1)`
- ✓ Pour récupérer toutes les lignes de résultat de requêtes (tuples):
`curseur.execute("""SELECT id_etud, nom FROM Etudiant""")`
`for L in curseur:`
`print( L)`
- ✓ Pour récupérer toutes les lignes de résultat de requêtes (liste):
`curseur.execute("""SELECT id_etud, nom FROM Etudiant""")`
`lignes = curseur.fetchall()`
`for L in lignes:`
`print(" id :", L[0], "nom: ", L[1]))`

RÉSUMÉ: SQLITE ET PYTHON (1)

1) Importer le module et se connecter à la BD

```
import sqlite3
```

```
conn=sqlite3.connect('exemple.db')
```

2) Créer un curseur afin d'exécuter des requêtes SQL

```
c = conn.cursor()
```

3) Passer en argument les requêtes à la méthode execute de l'objet cursor

```
c.execute("requête SQL")
```

```
c.executemany("requête SQL")
```

RÉSUMÉ: SQLITE ET PYTHON (2)

4) Afficher les résultats d'une requête SQL

- ✓ `Fetchone()` : résultat récupéré en une ligne unique (tuple)
- ✓ `Fetchmany(n)` : récupère les n prochains résultats (liste)
- ✓ `Fetchall()` : récupère toutes les lignes de résultat de la requete (liste)

5) Validation de modifications

`conn.commit()`

6) Annulation des dernières modifications

`conn.rollback()`

7) Fermer la connexion

`conn.close()`

EXAMPLE

```
import sqlite3
cnx=sqlite3.connect('baseTest.sq3')
cur=cnx.cursor()
cur.execute("""create table IF NOT EXISTS Etudiant(idEtud
INTEGER PRIMARY KEY autoincrement,Nom varchar(20),
Prenom varchar(20), AdresseText)""")
cur.execute(""" INSERT INTO Etudiant(Nom, Prenom) VALUES
("Arbi","moez ")""")
etud= []
etud.append(("Selmi","Ali"))
etud.append((" Arbi ", " Arwa"))
etud.append((" Arbi ", " Arij" ))
cur.executemany(""" INSERT INTO Etudiant(nom, prenom)
VALUES(?,?) """, etud)
```

EXAMPLE: SUITE.....

```
cur.execute("""SELECT nom, prenom FROM Etudiant""")
e1 = cur.fetchone()
print(e1)
cur.execute("""SELECT idEtud, nom,prenom FROM
etudiant""")
lignes = cur.fetchall()
for L in lignes:
    print("ID: ", L[0]," nom: ",L[1],"prenom: ", L[2])

cur.execute(""" DROP TABLE Etudiant """)

cnx.commit()
cnx.close()
```

LANGUAGE SQL:

Exemples de Bases de données(1)

22

Atef Boujelben

Exemple1:

Conference (nomC, universiteC, adresseC, dateC)

Participant(numeroP, nomP, prenomP, adresseP)

Organisateur (#numeroP, telephone, **#nomC**)

Conferencier (#numeroP, institution)

Inscrit (#numeroP, nomC#, droitinscription)

Exemple 2:

Classe (codeCl, libelle)

Etudiant (Num, nom, prenom, groupe, dateN, adr, ville, #codeCl)

LANGUAGE SQL:

Exemples de Bases de données (2)

23

Atef Boujelben

Exemple 3:

Table: Employer

EmpNo	nomE	prenomE	poste	datNai	Salaire	adr	DeptTNo
7369	Arbi	Ali	ouvrier	17-DEC-80	800.00	El Ain-Sfax	2
7499	Naifar	Sameh	chef	20-FEB-81	1600.00	Afran-Sfax	3
7521	Salhi	Samia	magazini	22-FEB-81	1250.00	Sokra-Sfax	3

Table: Département

<u>DepartNo</u>	nomD	Location
1	Idara	Sfax médina
2	Idara2	Sfax médina
3	Dépot	El Ain centre
4	Opération	Sekiit Ezit

Département (departNo, nomD, location)

Employer(empNo, nomE, prenomE, poste, dateNais, salaireE, adresseE ,# departNo)

CRÉATION DE TABLES: EXEMPLES

```
CREATE TABLE participant(  
  numeroP number(5) PRIMARY KEY,  
  nomP char(30) NOT NULL,  
  prenomP char(30),  
  addressP char(50) );
```

```
CREATE TABLE inscrit(  
  numeroP number(5), nomC char(30), droitinscription number(5),  
  constraint pri_key Primary Key (numeroP, nomC),  
  constraint for_key1 Foreign Key (numeroP) references  
  participant(numeroP),  
  constraint for_key2 Foreign Key (nomC) references  
  Conference(nomC) );
```

MODIFICATION.....

Modification de tables

```
ALTER TABLE nom_table  
ADD|MODIFY(attribut TYPE,...);
```

Exemple:

```
ALTER TABLE participant  
MODIFY(numeroP number(3));
```

Modification du nom de la table

```
ALTER TABLE orig_table_name RENAME TO tmp_table_name;
```

Suppression de contraintes

```
ALTER TABLE nom_table  
DROP CONSTRAINT nom_const;
```

MISE À JOUR D'UNE TABLE :

INSERTION/DESTRUCTION/ MODIFICATION

Insert into R(a1,a2, .., aN) values (v1,v2, .., vN);

Exemple :

Insert into Inscrit (numeroP, nomC, droitinscription) values
(‘153’,’JCC’, 300);

Delete from R Where condition;

Exemple :

Delete from Participant
where NumeroP = 1210;

Update R set a1=v1, a2=v2, ... aN=vN;

Exemple :

Update Conference set universite = ‘Sfax’;

SYNTAXE GÉNÉRALE D'UNE RECHERCHE

SELECT <liste des attributs>

FROM <listes des relations touchées par la question>

[**WHERE** <liste des critères de restriction>]

[**GROUP BY** <liste des attributs d'agrégation>]

[**HAVING** <liste des critères de restriction d'agrégats>]

[**ORDER BY** <liste des attributs de tri du résultat>]

[**Limit** nombre d'enregistrements souhaités];

LA REQUÊTE SELECT(1)

```
SELECT liste_d_attributs  
FROM liste_de_tables  
WHERE conditions;
```

projection

selection

SELECT indique la liste des attributs constituant le résultat

FROM le (les) table(s) utile(s) à la requête

WHERE indique les conditions que doivent satisfaire les tuples de la base pour faire partie du résultat

LA REQUÊTE SELECT(2)

Exemple1:

```
Select *  
from participant;
```

* remplace tous les tuples

Cette requête retourne tous les tuples de la table participant

Exemple 2 :

```
Select numeroP, nomP  
from participant  
where prenomP = 'Ali';
```

❑ Cette requête retourne le **numéro** et le **nom** de tous les participants ayant pour prénom 'Ali'

LES DOUBLONS

Contrairement à l'algèbre relationnelle, SQL
ne supprime pas les doublons

Exemple :

```
Select prenomP  
from participant
```

prenomP
Ali
Salah
Salma
Inès
Ali
Mohamed
Salah
Anis
Ali

Pour éviter d'avoir des tuples identiques
en utilise le mot clé **DISTINCT**

Exemple :

```
Select DISTINCT(prenomP)  
from participant;
```

PrenomP
Salah
Salma
Inès
Mohamed
Anis
Ali

TRIER LES RÉSULTATS

Il est possible de trier les données à l'aide de la clause **ORDER BY** suivie de la liste des attributs servant comme critère de tri. Par défaut l'ordre est ascendant (ASC), sinon on peut l'obliger de suivre un ordre descendant(DESC).

Exemple :

```
Select *  
from participant  
ORDER BY nomP, DESC
```

LA CLAUSE WHERE

Dans la clause WHERE, on spécifie une condition portant sur les attributs des relations mentionnées par la clause FROM.

Les opérateurs utilisés:

- ✓ AND, OR et NOT
- ✓ Les opérateurs de comparaison ($\leq$, $\geq$, $\neq$)
- ✓ BETWEEN pour les intervalles
- ✓ IN pour un ensemble de valeurs

Exemples:

```
select NomP  
from Inscrit  
where droitinscription BETWEEN 100 and 1000;
```

```
select NomP from Participant  
where prenomP IN ('Ali','Amine','Arij');
```

LES CHAÎNES DE CARACTÈRES

- ✓ SQL ne distingue pas entre les minuscules et les majuscules pour les mots clés. Ceci n'est pas vrai pour les valeurs, 'Tunisie' est différente de 'TUNISIE'
- ✓ SQL fournit des options pour les recherches par motif (pattern) à l'aide de la clause **LIKE**
 - '%' désigne n'importe quelle chaîne de caractères
 - '_' désigne une lettre

Exemples :

Select * from Conference where nomC LIKE '%a';
#Toutes les Conferences dont le nom finit par ' a '

Select * from Conference where nomC LIKE '___a%';
#Toutes les Conferences dont la quatrième lettre est ' a '

LES DATES

Dans SQL une date s'écrit au format 'DD-MON-YY'

Exemple :

```
Select nomC  
from Conference  
where dateC BETWEEN '03-NOV-23' AND '03-NOV-24'
```

LES VALEURS NULLES

- ✓ Les valeurs de certains attributs sont inconnus. Ils sont mis à NULL
- ✓ Toute comparaison avec NULL donne un résultat qui n'est ni vrai ni faux mais une troisième valeur UNKNOWN
- ✓ NULL est un mot clé et non pas une constante
 - TRUE 1
 - UNKNOWN 1/2
 - FALSE 0

LES REQUÊTES SUR PLUSIEURS TABLES

Les requêtes SQL décrites dans cette section permettent de manipuler simultanément plusieurs tables et d'exprimer des opérateurs d'algèbre relationnelle:

- ✓ Union,
- ✓ Intersection,
- ✓ Différence
- ✓ Produit cartésien
- ✓ Jointure.

UNION

L'union de 2 tables T1 et T2 produit une table T3 de même schéma contenant toutes les lignes appartenant à T1 ou à T2 ou aux 2 tables.

Exemple: donner les numéros et les noms des employés du département numéro 3 ainsi que les numéros et les noms des étudiants PC2.

```
Select empNo,nomP  
From employer  
Where deptNo=3
```

Union

```
Select Num,nom  
From Etudiant  
Where codeCl='PC2'
```

INTERSECTION: INTERSECT

L'intersection de 2 tables T1 et T2 produit une table T3 de même schéma contenant toutes les lignes appartenant à T1 et T2.

Exemple: donner les noms des étudiants de PC2 qui se répètent dans le groupe 1 et le groupe 2

```
Select nom  
From étudiant  
Where codeCl='PC2' and groupe=1
```

Intersect

```
Select nom  
From étudiant  
Where codeCl='PC2' and groupe=2
```

DIFFÉRENCE: EXCEPT

Exemple: donner les classes qui n'ont pas d'étudiants

```
Select codeCl  
From Classe
```

EXCEPT

```
Select codeCl  
From étudiant
```


PRODUIT CARTÉSIEN

Le produit cartésien est la concaténation des n-uplets des deux tables.

Exemple: donner les étudiants et les classes

```
Select *  
From étudiant,Classe
```

Remarque: pas de where dans le produit cartésien

JOINTURE DE BASE

Une jointure permet d'exprimer des requêtes portant sur des données réparties sur plusieurs tables.

Exemple: Donner le nom et le prénom de tous les participants du Conference Conference1

```
Select NomP, PrenomP
from Inscrit, Participant
where      Inscrit.NumeroP=Participant.NumeroP      and
NomC='Conference1';
```

✓ Comme ce n'est pas pratique de recopier intégralement le nom d'une table, on peut définir et utiliser des synonymes

```
Select NomP, PrenomP
from Inscrit as I, Participant as P
where I.NumeroP=P.NumeroP and NomC='Conference1';
```

JOINTURE EN SQL2

```
Select NomP, PrenomP  
from Inscrit INNER JOIN Participant  
ON Inscrit.NumeroP=Participant.NumeroP  
where NomC='Conference1';
```

Remarques:

- ✓ L'oubli de **ON** empêche l'exécution de la requête
- ✓ **INNER** est facultatif dans la plupart des SGBDR
- ✓ Cette notation rend plus lisible la requête en distinguant clairement les conditions de jointures, derrière ON, et les éventuelles conditions de sélection ou restriction, derrière WHERE.

JOINTURE ET ALIAS (1)

- ✓ Dans l'ambiguïté (deux attributs ayant le même nom), on met le nom de la table comme préfixe
- ✓ Comme ce n'est pas pratique de recopier intégralement le nom d'une table, on peut définir et utiliser des synonymes (alias).

Exemple:

```
Select NomP, PrenomP  
from Inscrit I, Participant P  
where I.NumeroP = P.NumeroP and NomC='Conference1';
```

OU BIEN:

```
Select NomP, PrenomP  
from Inscrit I JOIN Participant P ON I.NumeroP = P.NumeroP  
where NomC='Conference1';
```

JOINTURE ET ALIAS (2)

On peut faire intervenir plus que deux tables

Exemple:

```
Select NomP, PrenomP, DateC  
from Inscrit I, Participant P, Conference C  
where I.NumeroP = P.NumeroP and I.NomC = C.NomC;
```

OU BIEN:

```
Select NomP, PrenomP, DateC  
from Inscrit I JOIN Participant P ON I.NumeroP =  
P.NumeroP JOIN Conference C ON I.NomC = C.NomC;
```

OU BIEN:

```
Select NomP, PrenomP, DateC  
from Inscrit I JOIN Participant P JOIN Conference C  
ON I.NumeroP = P.NumeroP and I.NomC = C.NomC;
```

JOINTURE SUR LA MÊME TABLE

Exemple:

Donner des couples de Conférences organisés par la même université.

```
Select C1.NomC, C2.NomC  
From Conference C1, Conference C2  
Where C1.Universite = C2.Universite
```

Remarque : C1 et C2 sont 2 instances différentes de la table Conference

REQUÊTES IMBRIQUÉES(1)

- ✓ Le résultat d'une requête sert comme condition dans la deuxième requête
- ✓ Ces types de requêtes peuvent porter sur plusieurs tables
- ✓ Conditions portant sur des relations

Exemple:

```
Select numeroP from inscrit  
where nomC In ( select nomC from Conference  
                where Universite = 'Sfax');
```

Remarque: On peut utiliser '=' à la place de **IN** si on est sûr que la sous requête ramène **un et un seul tuple**

REQUÊTES IMBRIQUÉES(2)

✓ Sous requêtes corrélées

Exemple: Donner tous les participants ayant organisé au moins une fois et font une présentation dans une Conference

```
Select numeroP from organisateur O
where Exists (select * from conferencier C
              where O.numeroP = C.numeroP);
```

LES SOUS-REQUÊTES

- ✓ SQL permet d'utiliser des requêtes pour élaborer des conditions plus complexes et "dynamiques".

un critère de recherche employé dans la clause WHERE soit lui même le résultat d'un SELECT

- ✓ Types de sous-requêtes
 - ✓ Sous-requêtes simples
 - ✓ Sous-requêtes synchronisées

LES SOUS-REQUÊTES SIMPLES

✓ Dans ce cas la sous-requête est **d'abord évaluée, puis le résultat est utilisé** pour exécuter la requête principale.

Exemple: sélectionner la liste des employés ayant le même poste que ALI.

```
select nomE from Employer
where poste = ( select poste from Employer
               where upper(nomE) = 'ALI' );
```

Remarque: Les sous-requêtes sont plus lisibles que des jointures.

LES SOUS-REQUÊTES SYNCHRONISÉES

✓ Dans ce cas la sous-requête est évaluée pour chaque nuplet de la requête principale.

Exemple: sélectionner des employés ne travaillant pas dans le même département que leur manager.

```
Select nomE from Employer e
where departNo != ( select departNo from Employer
                    where e.empNo = empNo);
```

Remarque: le synonyme « e » de la requête principale est utilisé dans la sous-requête.

LES OPÉRATEURS ASSOCIÉS AUX SOUS REQUÊTES (1)

L'opérateur **EXISTS** permet de construire un prédicat vrai si la sous-requête qui suit ramène au moins une ligne.

Syntaxe:

```
select --- from nom-table where exists ( select --- )
```

LES OPÉRATEURS ASSOCIÉS AUX SOUS REQUÊTES (2)

Les opérateurs ensemblistes

- ❑ Nous avons travaillé avec le SELECT de la sous-requête ramenait un seul n-uplet, car à droite du signe "=" se trouvait une seule valeur. Cependant une sous-requête peut ramener plusieurs n-uplets (une liste de valeur).
- ❑ Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont les opérateurs obtenus en ajoutant **ANY** et **ALL** à la suite d'un opérateur de comparaison:
 - ✓ **ANY** : la comparaison sera vraie si elle est vraie pour au moins un élément de l'ensemble
 - ✓ **ALL** : la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble

LES OPÉRATEURS ASSOCIÉS AUX SOUS REQUÊTES (3)

Remarques:

L'opérateur **IN** est équivalent à l'opérateur **= ANY**

L'opérateur **NOT IN** est équivalent à l'opérateur **!= ALL**

✓ Les sous-requêtes ramenant plusieurs colonnes

- Il est possible de comparer le résultat d'un **SELECT** ramenant plusieurs colonnes à une liste de colonnes.
- La liste de colonnes figurera entre parenthèses à gauche de l'opérateur de comparaison.

Syntaxe:

```
select col1, ... from nom_table
```

```
where (col2,col3,...) = (select col2, col3,... ---)
```

LES FONCTIONS DE GROUPES(1)

- ✓ Il est possible d'utiliser certaines fonctions en demandant de grouper les résultats selon une ou plusieurs colonnes.
- ✓ Les fonctions **d'agrégation** permettent de faire des opérations sur une sélection de champ. Elles effectuent un calcul sur un ensemble de valeurs et retournent une valeur unique.
- ✓ À l'exception de COUNT, les fonctions d'agrégation ignorent les valeurs NULL.
- ✓ Les fonctions d'agrégation sont souvent utilisées avec la clause GROUP BY de l'instruction SELECT.

LES FONCTIONS DE GROUPES(2)

✓ Les fonctions de groupe sont :

- **avg (col)** : moyenne des valeurs (les valeurs NULL sont ignorées)
- **count (col)**: nombre de n-uplet satisfaisant à la condition WHERE.
- **max (col)**: valeur maximale des valeurs de la colonne
- **min (col)**: valeur minimale des valeurs de la colonne
- **sum (col)**: somme des valeurs de la colonne
- **variance (col)**: variance des valeurs de la colonne

EXAMPLES

```
Select count (NomC)  
From Conference;
```

```
Select sum (droitinscription)  
From inscrit Where NumeroP = 100;
```

```
Select count(numStation), AVG (tarif), MAX (tarif), MIN (tarif)  
From Station;
```

Remarque: dans ces exemples, il n'y pas d'attribut dans select, sinon on doit faire groupe by

CALCUL SUR PLUSIEURS GROUPES

Il est possible de subdiviser la table en groupes, chaque groupe étant l'ensemble des lignes ayant une valeur commune.

Syntaxe

group by expr [,expr2 ...]

Exemple: donner la somme des salaires pour chaque département.

```
select DepartNo, SUM(salaireE)
from Employer
group by DepartNo;
```

REGROUPEMENT: CLAUSE SELECT

Dans un regroupement la clause SELECT a deux parties:

- ✓ liste des attributs servant au regroupement
- ✓ liste des fonctions ensemblistes évaluées sur chaque groupe



regroupement **fonctions**

```
SELECT NO_CLIENT, SUM(MONTANT)
FROM COMMANDE
GROUP BY NO_CLIENT;
```

REGROUPEMENT: CLAUSE HAVING

- ✓ La clause HAVING permet, de façon analogue à la clause WHERE, de poser des conditions sur le résultat du regroupement

```
select col1 [,col2 ...]  
from table1 [, table2]  
[where prédicat]  
[group by expression1 [, expression2...]  
[having prédicat] ];
```

- ✓ Le prédicat figurant dans la clause HAVING ne peut porter que sur des caractéristiques des fonctions de groupe figurant dans la clause GROUP BY

LA CLAUSE GROUP BY/ HAVING

✓ La clause GROUP BY Construit des groupes en associant les tuples partageant la même valeur pour une ou plusieurs colonnes

Exemple:

```
Select universite, count (NomC)
from Conference
Group by universite;
```

✓ On peut ajouter une condition sur le groupe

Exemple:

```
Select universite, count (NomC) From Conference
Group by universite
Having count (NomC) >= 3;
```

LA CLAUSE LIMIT

La clause **LIMIT** est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'on souhaite obtenir.

Exemple: Donner le nom et la marque des deux plus chers produits.

```
SELECT P.nom, P.marque  
FROM produit P  
Order by P.prix DESC  
Limit 2
```

SQL AS (ALIAS)

Dans le langage SQL il est possible d'utiliser des **alias** pour renommer temporairement une colonne ou une table dans une requête. Cette astuce est particulièrement utile pour faciliter la lecture des requêtes.

Intérêts et utilités: Alias sur une colonne

Permet de renommer le nom d'une colonne dans les résultats d'une requête SQL. C'est pratique pour avoir un nom facilement identifiable dans une application qui doit ensuite exploiter les résultats d'une recherche.

```
Select universite, count (NomC) as nombre_Conf
From Conference
Group by universite
Having nombre_Conf >= 3;
```

SQL AS (ALIAS)

Intérêts et utilités: Alias sur une table

La syntaxe pour renommer une table dans une requête est la suivante:

```
SELECT *  
FROM 'nom_table' AS t1
```