CHAPITRE 6: LANGAGE SQL

I. Quelques rappels

Les données constituant la base sont organisées en tables. Une table contient des enregistrements composés de différents champs. Le type associé au champ défini le domaine sur lequel le champ pourra prendre ses valeurs (date, entier, chaîne de caractères).

Chaque table est désignée de manière unique par un identificateur (son nom) au sein de la base de données de même que chaque champ au sein d'une table.

Usuellement on représente une table par un tableau dont les colonnes correspondent aux champs et les lignes aux enregistrements :

La clé primaire d'une table est un champ (ou un ensemble de champs) qui identifie de manière unique chaque enregistrement dans la table. Chaque table devrait avoir une clé primaire.

Une clé étrangère dans une table est un champ (ou un ensemble de champs) qui fait référence à un champ (ou un ensemble de champs) d'une autre table (généralement la clé primaire).

Les requêtes permettent d'interroger une base de données et d'en modifier les informations.

II. Qu'est-ce que SQL ?

Le **SQL** est un langage de requête structuré (***Structured Query Language***) destiné à communiquer avec des bases de données relationnelles implémentées dans des *SGBDR* (Système de Gestion de Bases de Données Relationnelles) tels que **SQLite**, Oracle, Access, SQL Server ou encore Sybase.

Le langage **SQL** permet d'effectuer diverses opérations comme la création, l'extraction, la modification, la suppression, la fusion, etc., sur des collections de données, par l'intermédiaire d'instructions particulières appelées des commandes, souvent assistées d'ailleurs par des clauses ou des options.

La première mouture du langage **SQL** est apparu en 1979, puis sera standardisé à quatre reprises; en 1986 avec le **SQL-ANSI** (American National Standard Institute), en 1989 avec **SQL-ISO** (International Standard Application) et ANSI, en 1992 avec la seconde version du **SQL-ISO** et **ANSI** et enfin en 1999, pour établir la troisième version.

III. Les commandes SQL :

Les commandes **SQL** se divisent en trois catégories principales pour : la définition, la manipulation, l'interrogation des données :

- **Le langage de définition** : propose des commandes de création (***CREATE***), suppression (***DROP***) et modification (***ALTER***) de tables, d'index, de contraintes, etc...
- **Le langage de Manipulation :** permet d'insérer (***INSERT***), de mettre à jour (***UPDATE***) des enregistrements, de supprimer (***DELETE***) des données, mais également de manipuler des curseurs (***DECLARE***, ***OPEN***, ***FETCH***, ***CLOSE***) au sein d'une base de données relationnelles.

- **Le langage de requête :** fournit une commande de sélection d'enregistrements (***SELECT***) qui avec ses très nombreuses clauses et options offrent un support d'interrogation des bases de données très puissant et complet.

Il existe de nombreuses autres catégories de commandes comme celles explicitées ci-dessous.

- **Le langage de contrôle :** est utilisé pour la gestion des droits d'accès aux utilisateurs en attribuant (***GRANT***) ou révoquant (***REVOKE***) des privilèges.
- **Les commandes d'administration :** de données sont utilisées pour la réalisation d'audits et d'analyses d'opérations sur des bases de données.
- **Les commandes transactionnelles :** permettant de gérer les transactions de base de données. Il s'agît de notamment de ***COMMIT***, ***ROLLBACK***, ***SAVEPOINT*** et ***SET TRANSACTION***.

IV. Les commandes DDL:

Les commandes **DDL** (***Data Defintion Language***) définissant la structure d'une base de données, permet la création, la modification ou la suppression de divers objets tels qu'une table, une vue ou encore un index.

IV-1. La commande CREATE TABLE :

Une base de données relationnelle est composée de tables ou relations constituant un ensemble logique. Chaque table doit être créée par la commande ***CREATE TABLE*** : elle reçoit ainsi un nom et une structure (liste de noms d'attributs et quelques spécifications). La table existe physiquement mais elle est encore vide de données. Elle n'est alors qu'une structure logique capable de recevoir des données

Syntaxe :

```
CREATE TABLE table
(champ type CONSTRAINT champ propriétés, ... );
```

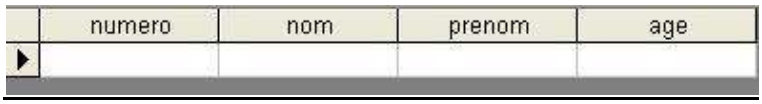
Champ	Nom du champ (Attribut)
Type	Type de données, dans la plupart des versions de SQL, vous aurez droit aux types de données suivants : • **Char(x)** : chaîne de caractères, x est le nombre maximum de caractères autorisés dans le champ. • **Integer** : Nombre entier, positif ou négatif • **Decimal (x,y)** : Nombre décimal, x est le nombre maximum de chiffres et y le nombre maximum de chiffres après la virgule. **Rq** *: Decimal ne fonctionne pas avec Access, il ne supporte que le type 'float' (flottant), le type float ne permet pas d'indiquer le nombre de chiffres après ou avant la virgule* • **Date** : Une date et/ou heure • **Logical** – Deux valeurs possibles : oui / non
propriétés	Propriétés du champ : • **NULL** ou **NOT NULL** : autorise ou non que le champ puisse être vide. • **UNIQUE** : indique que deux enregistrements ne pourront avoir la même valeur dans ce champ.

	• **PRIMARY KEY** : indique que ce champ est la clef primaire • **CHECK (condition)** : va forcer SQL a faire une vérification de la condition avant de valider la saisie, exemple : CHECK (prix > 100) interdira la saisie dans ce champ si la valeur contenue dans le champ prix est inférieure à 100. **Rq** *:* CHECK ne fonctionne pas avec Access. • **DEFAULT = valeur** : place une valeur par défaut dans le champ **Rq** :ne fonctionne pas avec Access)

Exemple : La commande suivante crée la table " Personnes ", contenant 3 attributs :

```
CREATE TABLE Personnes
(numero integer , nom CHAR(20) NOT NULL,
prenom CHAR(20),age integer)
```

Engendre la création de la table ci-dessous :



La clé primaire:

Créons, par exemple, la table "Personnes" avec une clé primaire sur le champ "numero". La commande s'écrit :

```
CREATE TABLE Personnes
(numero integer PRIMARY KEY , nom CHAR(20) NOT NULL,
prenom CHAR(20),age integer)
```

Pour appliquer la clé à deux champs, nous utilisons la syntaxe suivante :

```
CREATE TABLE Personnes
   (numero integer , nom CHAR(20) NOT NULL,
   prenom CHAR(20),age integer,
   PRIMARY KEY(numero, Nom));
```

La clé étrangère:

Supposant qu'on a les deux tables suivantes :

- Elève(**CNE**,Nom,Prénom,#NClasse)
- Classe(**NClasse**,Nbr_élèves)

Créons la table "Elève" avec une clé primaire sur le champ "CNE" et la clé étrangère sur le champ " NClasse" La commande s'écrit :

```
CREATE TABLE Elève
(CNE integer PRIMARY KEY , Nom CHAR(20) , Prénom
CHAR(20), NClasse integer,
FOREIGN KEY (NClasse) REFERENCES Classe (NClasse))
```

IV-2. La commande ALTER TABLE :

La commande ***ALTER TABLE*** est une requête permettant la modification d'une table en ajoutant, en supprimant ou en changeant les colonnes, leur définition ou les contraintes.

Syntaxe :

- ajoute les colonnes spécifiées à une table existante.

ALTER TABLE Nom_*table* **ADD** *colonne type*

- supprime la colonne *col.*

ALTER TABLE Nom_*table* **DROP** *colonne*

- modifie la définition des colonnes spécifiées.

ALTER TABLE Nom_*table* **MODIFY** *colonne type*

Exemple : Il est possible de modifier une table existante. Les exemples les plus classiques concernent l'addition d'une nouvelle colonne et la suppression d'une colonne existante. La commande :

```
ALTER TABLE Personnes
ADD Naissance DATE
```

Permet, lorsqu'on l'exécute, d'ajouter le champ intitulé "Naissance", de type Date/Heure, à la table "Personnes". La variante suivante fonctionne également :

```
ALTER TABLE Personnes
ADD COLUMN Naissance DATE
```

Pour supprimer la colonne que nous venons de créer, nous utilisons la commande suivante :

```
ALTER TABLE Personnes
DROP Naissance
```

Ou sa variante :

```
ALTER TABLE Personnes
DROP COLUMN Naissance
```

En SQL standard, la commande ***ALTER TABLE*** peut aussi être utilisée pour modifier les propriétés d'une colonne existante.

```
ALTER TABLE Personnes
MODIFY Nom CHAR(40)
```

Remarque :

La clause ***MODIFY*** n'est pas reconnue par Access, et l'exécution de la commande ci-dessus entraîne un message d'erreur. L'ignorance de la clause MODIFY enlève à la commande ***ALTER TABLE*** une bonne partie de son intérêt dans Access.

IV-3. La commande DROP TABLE :

La commande ***DROP TABLE*** est une requête permettant la suppression complète d'une table.

Syntaxe :

```
DROP TABLE Nom_Table
```

Exemple :

```
DROP TABLE Personnes
```

V. Les commandes DML:

Les commandes ***DML (Data Manipulate Language)*** permettent de manipuler les informations d'une base de données.

Il existe trois commandes destinées a mettre à jour (***UPDATE***), insérer (***INSERT***) ou supprimer (***DELETE***) des données.

V-1. La commande INSERT INTO :

La commande ***INSERT INTO*** permet d'ajouter des données dans une table.

Syntaxe :

```
INSERT INTO Nom_Table
        (Champ_1, Champ_2, ..., Champ_N)
  VALUES (Valeur_1, Valeur_2, ..., Valeur_N) ;
```

Les valeurs de colonnes doivent être encadrées par des guillemets simples (') s'il s'agît de chaîne de caractères, les nombres ne nécessitant pas de guillemets.

Exemple :

```
INSERT INTO personnes
        (numero,nom,prenom,age)
          VALUES (1, "Bennani ", "Taj", 18) ;
```

Il est possible de ne pas énumérer les noms de colonnes si l'ajout de données concerne un enregistrement complet de la table.

```
INSERT INTO Nom_Table
    VALUES (Valeur_1, Valeur_2, ..., Valeur_N)
```

En général, la citation des noms de colonnes est pertinente pour un ajout de données limité à certain champs de la table.

```
INSERT INTO Nom_Table
            (Champ_1, Champ_2)
    VALUES (Valeur_1, Valeur_2)
```

L'insertion de valeur nulle peut se faire par l'intermédiaire du mot-clé **NULL** ou d'une chaîne vide.

```
INSERT INTO Nom_Table
        (Champ_1, Champ_2)
    VALUES (Valeur_1, NULL)

INSERT INTO Nom_Table
        (Champ_1, Champ_2)
    VALUES (Valeur_1, '')
```

V-2. La commande UPDATE :

La commande ***UPDATE*** permet de mettre à jour des données existantes au sein d'une table.

Syntaxe :

```
UPDATE Nom_Table
SET  Col_1= Nouv_Val_1, Col_2= Nouv_Val_2,…, Col_N= Nouv_Val_N
WHERE Condition
```

La commande ***UPDATE*** peut mettre à jour les données de manières différentes :

- **La valeur d'une seule colonne** en ne citant qu'une colonne associée à sa nouvelle valeur.
- **Plusieurs colonnes simultanément** en citant plusieurs colonnes avec leur valeur respective.
- Par l'intermédiaire d'une clause conditionnelle affectant quelques lignes de données, **les valeurs de plusieurs enregistrements en même temps.**

Exemple :

```
UPDATE Personnes
    SET age = 15
    WHERE Nom = "Bennani" AND Prenom = "Taj"
```

V-3. La commande DELETE FROM :

La commande ***DELETE FROM*** permet de supprimer des enregistrements au sein d'une table.

Syntaxe :

```
DELETE FROM Nom_Table
        WHERE Condition
```

La clause conditionnelle ***WHERE*** détermine les enregistrements à sélectionner pour effectuer leur suppression complète de la table.

Exemple :

```
DELETE FROM Personnes
        WHERE Nom = "Bennani"
```

Lorsque la clause ***WHERE*** est absente, le ***SGBD*** supprime tous les enregistrements, laissant la table vide (mais ne la supprimant pas) :

```
DELETE FROM Personnes;
```

VI. Interrogation d'une base :

COMMANDES	COMMENTAIRE	EXEMPLES
SELECT	Opérateur de projection, permettant de choisir les colonnes à afficher Suivi de noms de colonnes ou de * (indiquant toutes les colonnes)	Affiche le nom et le prénom de tous les élèves **SELECT NomE, PrenomE** **FROM ELEVES**
FROM	Suivi d'une ou plusieurs noms de table. La présence de plusieurs tables réalise le produit cartésien de ces tables.	Affiche tous les articles (toutes les colonnes) **SELECT *** **FROM Article**
WHERE	Opérateur de sélection, qui permet de choisir des lignes selon certains critères (séparés par **AND** ou **OR**) Cet opérateur permet aussi de réaliser des jointures, le plus souvent par égalité des deux colonnes de sens identique dans les deux tables. Critère de jointure Critère de sélection	Affiche la designation de tous les articles de la commande numéro 1258 **SELECT art_designation** **FROM Article, Commande** **WHERE Article.art_num = Commande.art_num** **AND num_com = 1258**
DISTINCT	La clause DISTINCT permet d'éliminer les doublons : si dans le résultat plusieurs lignes sont identiques, une seule sera conservée	Afficher toutes les villes où habite au moins un élève **SELECT DISTINCT VilleE** **FROM ELEVES**
ORDER BY	La clause ORDER BY permet de trier les résultats par ordre croissant (ASC) ou décroissant (DESC). Le tri peut se faire sur une ou plusieurs colonnes	Afficher la liste des employés par salaire décroissant **SELECT NomEmp, sal FROM EMPLOYE** **ORDER BY sal DESC** Afficher les élèves par ordre

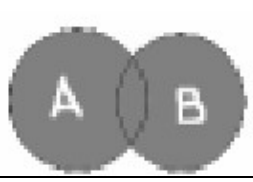
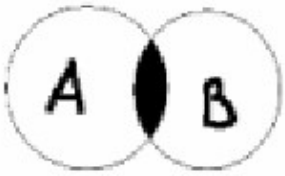
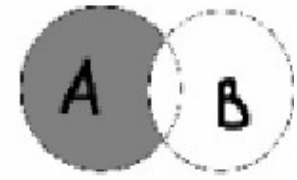
	L'option ASC est prise par défaut pour chacune des expressions citées	alphabétique du nom puis du prénom **SELECT * FROM ELEVES** **ORDER BY NomE, PrenomE**
'	Les cotes sont obligatoires pour entourer les chaînes de caractère et les dates. Sur Access on utilise '' pour les chaînes et # # pour les dates	Recherche tout les articles de couleur rouges : **SELECT art_num FROM Article** **WHERE art_coul = ' ROUGE'** **SELECT NumE FROM ELEVES** **WHERE ddn = '12/10/1980'**
IN	IN indique si la valeur est égale à l'une de celles qui se trouve entre parenthèse On peut utiliser aussi NOT IN (la valeur n'est égale à aucune de celles de la liste)	Liste des employés occupant la fonction de programmeur, analyste ou développeur **SELECT NomEmp , NumEmp FROM EMPLOYE** **WHERE fonction IN ('programmeur', 'analyste',** **developpeur');**
NULL	Indique une valeur non définie (pas entrée). Attention, 0 n'est pas une valeur NULL L'opérateur de comparaison est IS ou IS NOT	Afficher le nom des professeurs dont on ne connaît pas le prénom **SELECT NomP** **FROM PROFS** **WHERE PrenomP IS NULL**
LIKE	Permet de comparer une valeur avec une chaîne non complète % désigne plusieurs caractères _ désigne un seul caractère quelconque (ou aucun) Sous Access on utilise * à la place de %	Recherche tout les noms commençant par DE : **SELECT NomC, PrenomC, RueC** **FROM COMMERCANTS** **WHERE NomC LIKE 'DE% '**
BETWEEN	Indique si une valeur est comprise entre deux valeurs. Les extrêmes sont inclus dans l'intervalle.	Articles qui coutent entre 10 et 20 € inclus. **Select art_num, art_lib** **FROM Article** **WHERE prix BETWEEN 10 AND 20**
OPERATEURS DE CALCUL : **+ - * …**	Permet d'afficher le résultat d'un calcul à partir d'une ou plusieurs colonnes	Affiche la référence et la marge de tous les produits **SELECT refprod, (prix_vente –** **prix_achat) as marge** **FROM PRODUIT**

OPERATEURS D'AGREGATION

COMMANDES	COMMENTAIRE	EXEMPLES
COUNT (*)	Permet de compter le nombre de lignes résultant d'un résultat. On peut nommer la colonne correspondante avec AS.	Compte le nombre d'élève habitant Tanger et appelle le résultat Nb_Tanger. **SELECT COUNT (*) AS Nb_Tanger** **FROM ELEVES** **WHERE VilleE = 'Tanger'**
SUM (colonne)	Permet d'additionner les valeurs d'une	Calcul le cumul (la somme) de tous les

	colonne numérique pour les lignes sélectionnées	opérations de débit du compte 1259 le 09/01/04 **SELECT SUM(montant_opération) AS débit_0901** **FROM OPERATIONS** **WHERE compte = '1259'** **AND date = '09/01/04'**
AVG (colonne)	Permet d'afficher la moyenne des valeurs d'une colonne numérique pour les lignes sélectionnées	Afficher le salaire moyen des analystes **SELECT AVG(salaire)** **FROM EMPLOYE** **WHERE fonction = 'analyste'**
MAX (colonne), MIN (colonne	Permet d'obtenir la valeur maximale (ou minimale) d'une colonne pour un ensemble de lignes sélectionnées	Affiche le salaire de la secrétaire la mieux payée: **SELECT MAX(salaire)** **FROM EMPLOYE** **WHERE function = "secrétaire"**

OPERATEURS ENSEMBLISTES

COMMANDES	COMMENTAIRE	EXEMPLES
UNION	Il est possible de faire l'union des résultats de deux requêtes. L'union élimine les doublons On a alors : A ∪ B 	La liste des villes où habitent des élèvent et des professeurs : **SELECT VilleE** **FROM ELEVES** **UNION** **SELECT VilleP** **FROM PROFS**
INTERSECT	Il est possible de faire l'intersection des résultats de deux requêtes. On a alors : A ∩ B 	La liste des villes où habitent à la fois des élèves et des professeurs : **SELECT DISTINCT VilleE** **FROM ELEVES** **INTERSECT** **SELECT DISTINCT VilleP** **FROM PROFS**
EXCEPT ou MINUS	Il est possible de faire la différence des résultats de deux requêtes. On a alors : A – B 	La liste des villes où habitent seulement des élèves et pas de professeurs : **SELECT DISTINCT VilleE** **FROM ELEVES** **EXCEPT** **SELECT DISTINCT VilleP** **FROM PROFS**

VII. Interroger une base de données depuis un programme écrit en Python:

Nous allons voir sur un exemple comment effectuer une requête SQL depuis un programme écrit en Python et comment on récupère le résultat.

```
import sqlite3                          # importer le module sqlite3
con=sqlite3.connect('entreprise.sqlite')    #ouverture de la base de données 'entreprise.sqlite'
c=con.cursor()                          #obtention d'un curseur
r='select * from client'                    #Préparer la requête
c.execute(r)                            # exécution de la requête
#affichage des enregistrements récupérés par fetchall()
L=c.fetchall()
print(L)

for x in L:
        print(x[0], x[1],x[2])
con.close()                             #fermeture de la base
```

Résultat de ce code :

```
[ (1, 'Lamrani', 'najib', '1980-01-01 00:00:00', 'sifa', 10000, 'Rabat'), (2, 'Talbi', 'mohamed', '1981-07-06 00:00:00', 'rafaoui', 80000, 'Sidi kacem'), (3, 'maradani', 'omar', '1977-06-04 00:00:00', 'myalichrif', 80000, 'Rabat'), (4, 'farid', 'omar', '1979-06-03 00:00:00', 'atlass', 70000, 'Sidi Slimane'), (5, 'bouthir', 'abdelmajid', '1980-09-19 00:00:00', 'boustoh', 80000, 'Sidi kacem'), (6, 'boubekri', 'mohamed', '1979-10-22 00:00:00', 'boudir', 60000, 'Tanger'), (7, 'albakali', 'nadia', '1989-12-03 00:00:00', 'enakhil', 80000, 'Sidi kacem'), (8, 'lasri', 'fatima', '1963-06-11 00:00:00', 'rabat', 93150, 'Tetouan'), (9, 'chuichi', 'mohamed', '1981-06-23 00:00:00', 'sidi youssef', 90000, 'Sidi Slimane'), (10, 'chuichi', 'najib', '1978-08-23 00:00:00', 'lbour', 50000, 'Kenitra'), (11, 'kacha', 'najib', '1963-01-06 00:00:00', 'ghandi', 90000, 'Tanger')]
1 Lamrani najib
2 Talbi mohamed
3 maradani omar
4 farid omar
5 bouthir abdelmajid
6 boubekri mohamed
7 albakali nadia
8 lasri fatima
9 chuichi mohamed
10 chuichi najib
11 kacha najib
```