



SQL – Structured Query Language

Filière : Achats et Supply Chain Management - Master d'État

Par : Youssef Ouassit

- Introduction
- Langage de définition des données
- Langage de manipulation des données
- Exercices

- SQL : Structured Query Language
- Un langage normalisé servant à exploiter des bases de données
- Créé en 1974, normalisé depuis 1986
- Dernière révision en 2019
- Supporté par tous les SGBD :
 - MySQL
 - SqlServer
 - Oracle
 - MS Access
 - ...
- SQL n'est pas un langage de programmation

Le langage SQL comporte :

- une partie sur la définition des données :
Le langage de définition des données (LDD) qui permet de définir les relations, des vues et des contraintes d'intégrité
- une partie sur la manipulation des données :
Le langage de manipulation des données (LMD) qui permet d'interroger une base de données sous forme déclarative sans se préoccuper de l'organisation physique des données
- une partie sur le contrôle des données :
Le langage de contrôle des données (LCD) qui permet de contrôler la sécurité et l'accès aux données

LDD : Langage de Définition des Données

Avec le langage LDD on peut :

- Créer une base de données
- Créer une relation (une table)
- Supprimer une table
- Modifier la structure d'une table

Création d'une base de données

Syntaxe :

```
CREATE DATABASE NOM_DE_BASE;
```

Exemple :

```
CREATE DATABASE ECOLE;
```

Création des tables : Syntaxe

```
CREATE TABLE nom_de_relation  
(  
    nom_colonne type_de_données [AUTO_INCREMENT] [NOT NULL] [DEFAULT value],  
    ...  
  
    PRIMARY KEY (nom_colonne, ...) ,  
    FOREIGN KEY (nom_colonne) REFERENCES nom_autre_table (nom_autre_colonne) ,  
    ...  
)
```

Création des tables : Types

- INTEGER
- REAL
- TEXT
- DATE
- DATETIME
- BLOB

Création des tables : Exemple

```
CREATE TABLE Classe
(
    Id INTEGER AUTO_INCREMENT,
    Nom TEXT,
    PRIMARY KEY(Id)
)
CREATE TABLE Eleve
(
    Num INTEGER AUTO_INCREMENT,
    Nom TEXT,
    Prenom TEXT,
    Photo BLOB,
    Id_Classe INTEGER,
    PRIMARY KEY(Num),
    FOREIGN KEY (Id_Classe) REFERENCES Classe ( Id )
)
```

Classe
<u>Id</u> : Entier
Nom : Chaîne

Eleve
<u>Num</u> : Entier
Nom : Chaîne
Prenom : Chaîne
Photo : Fichier
Id_Classe* : Entier

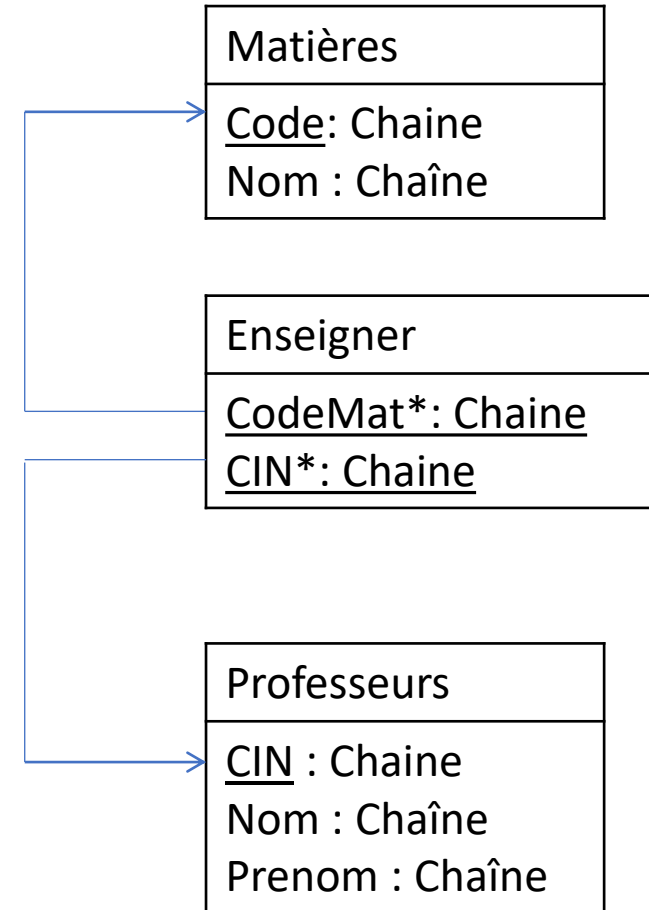


Création des tables : Exercice

```
CREATE TABLE Matières
(  
    Code TEXT,  
    Nom TEXT,  
    PRIMARY KEY(Code)  
)
```

```
CREATE TABLE Professeurs
(  
    CIN TEXT,  
    Nom TEXT,  
    Prenom TEXT,  
    PRIMARY KEY(CIN)  
)
```

```
CREATE TABLE Enseigner
(  
    CodeMat TEXT,  
    CIN TEXT,  
    PRIMARY KEY(CodeMat, CIN),  
    FOREIGN KEY (CodeMat) REFERENCES Matières ( Code ),  
    FOREIGN KEY (CIN) REFERENCES Professeurs ( CIN ),  
)
```



Modification et Suppression des tables

- Ajouter une colonne à une table :

```
ALTER TABLE nom_table ADD nom_colonne type_colonne
```

- Supprimer une colonne d'une table :

```
ALTER TABLE nom_table DROP COLUMN nom_colonne
```

- Supprimer une table :

```
DROP TABLE nom_table
```

LMD : Langage de Manipulation des Données

Avec le langage LMD on peut :

- Insérer des données dans une table
- Modifier les données d'une table
- Supprimer les données d'une table
- Sélectionner des données d'une ou plusieurs tables

Insertion des données dans une table

- Insérer les valeurs de toutes les colonnes :

```
INSERT INTO nom_table VALUES(val1, val2, val3, ...)
```

- Insérer les valeurs d'une parties de colonnes :

```
INSERT INTO nom_table(colonne1, colonne2, ...) VALUES (val1, val2, ...)
```

Insertion des données dans une table

Exemple :

```
CREATE TABLE Eleve(  
    Num INETEGR AUTO_INCREMENT NOT NULL,  
    Nom TEXT NOT NULL,  
    DateNaissance DATE,  
    Actif TEXT NOT NULL DEFAULT 'Oui';  
)
```

Eleve
<u>Num</u> : Entier
Nom : Chaîne
DateNaissance:
Date
Actif : Chaîne

Insertion des données dans une table

Exemples :

- **Insérer les valeurs de toutes les colonnes :**

```
INSERT INTO Eleve VALUES(1, 'Nadine', '11-01-1990','Oui')
```

- **Insérer les valeurs d'une parties de colonnes :**

```
INSERT INTO Eleve (Num, Nom) VALUES (2, 'Malak')
```

```
INSERT INTO Eleve (Nom,Actif) VALUES ('Aziz','Non')
```

```
INSERT INTO Eleve (Nom) VALUES ('Ihssan')
```

Num	Nom	DateNaissance	Actif

Modification des données d'une table

- Syntaxe :

UPDATE **nom_table**

SET **colonne_1** = 'valeur 1', **colonne_2** = 'valeur 2' , ...

[WHERE **condition**]

- Exemples :

Rendre tous les élèves actifs :

UPDATE Eleve **SET** Actif = 'Oui'

Rendre l'élève numéro 2 actif :

UPDATE Eleve **SET** Actif = 'Oui' **WHERE** Num = 2

Augmenter l'âge des élèves actifs par 1 an :

UPDATE Eleve **SET** Age = Age + 1 **WHERE** Actif = 'Oui'

Eleve
<u>Num</u> : Entier
Nom : Chaîne
DateNaissance:
Date
Actif : Chaîne
Age : Entier

Suppression des données d'une table

- Syntaxe :

DELETE FROM nom_table [WHERE condition]

- Exemples :

Supprimer les élèves non actifs :

DELETE FROM Eleve WHERE Actif='Non'

Supprimer les élèves âgés plus que 23 ans :

DELETE FROM Eleve WHERE Age >=23

Supprimer tous les élèves :

DELETE FROM Eleve

TRUNCATE TABLE Eleve

Eleve
<u>Num</u> : Entier
Nom : Chaîne
DateNaissance: Date
Actif : Chaîne
Age : Entier

Sélection des données d'une base de données

- La requête SELECT :

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

De l'algèbre relationnel à SQL

- Forme de base de la commande SELECT :

SELECT < liste des colonnes de la table résultat >

FROM < liste des tables impliquées dans l'interrogation >

[WHERE < condition de sélection des lignes >]

[ORDER BY < attribut de tri >]

Projection des données :

Projeter une colonne :

Eleves

num	prenom	nom
1	Karim	Biyad
2	Nadia	Lahby
3	Karim	Kourchi
4	Meryem	Namir
5	Karim	Niya

SELECT prenom **FROM** Eleves

prenom
Karim
Nadia
Karim
Meryem
Karim

Projection des données :

Projeter plusieurs colonnes :

Eleves

num	prenom	nom
1	Karim	Biyad
2	Nadia	Lahby
3	Karim	Kourchi
4	Meryem	Namir
5	Karim	Niya

```
SELECT prenom,nom FROM Eleves
```

prenom	nom
Karim	Biyad
Nadia	Lahby
Karim	Kourchi
Meryem	Namir
Karim	Niya

Projection des données :

Projeter sans répétition:

Eleves

num	prenom	nom
1	Karim	Biyad
2	Nadia	Lahby
3	Karim	Kourchi
4	Meryem	Namir
5	Karim	Niya

```
SELECT distinct prenom FROM Eleves
```

prenom
Karim
Nadia
Meryem

Projection des données :

Projeter toutes les colonnes :

Eleves

num	prenom	nom
1	Karim	Biyad
2	Nadia	Lahby
3	Karim	Kourchi
4	Meryem	Namir
5	Karim	Niya

SELECT * FROM Eleves

num	prenom	nom
1	Karim	Biyad
2	Nadia	Lahby
3	Karim	Kourchi
4	Meryem	Namir
5	Karim	Niya

Sélection des données :

SELECT < liste des colonnes de la table résultat >

FROM < liste des tables impliquées dans l'interrogation >

WHERE < condition de sélection des lignes >

Sélection des données :

Eleves

num	prenom	nom
1	Karim	Biyad
2	Nadia	Lahby
3	Karim	Kourchi
4	Meryem	Namir
5	Karim	Niya

```
SELECT * FROM Eleves  
WHERE prenom = 'Karim'
```

num	prenom	nom
1	Karim	Biyad
3	Karim	Kourchi
5	Karim	Niya

Sélection des données :

Eleves

num	prenom	nom
1	Karim	Biyad
2	Nadia	Lahby
3	Karim	Kourchi
4	Meryem	Namir
5	Karim	Niya

```
SELECT * FROM Eleves  
WHERE prenom = 'Nadia'
```

num	prenom	nom
2	Nadia	Lahby

Sélection des données :

Les opérateurs de comparaison :

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

Sélection des données :

Liste des personnes n'ayant pas un numéro de téléphone :

SELECT * FROM Personnes WHERE telephone is NULL

Liste des personnes ayant un numéro de téléphone :

SELECT * FROM Personnes WHERE telephone is not NULL

Liste des personnes ayant un nom qui commence par A :

SELECT * FROM Personnes WHERE nom like 'A%'

Liste des personnes âgées entre 20 et 21 ans :

SELECT * FROM Personnes WHERE age between 20 and 21

Liste des personnes âgées de 19 ou 21 ou 22 ans :

SELECT * FROM Personnes WHERE age=19 or age=21 or age=22

SELECT * FROM Personnes WHERE age IN (19, 21, 22)

Liste des noms et téléphones des personnes âgées de 21 ans :

SELECT nom, telephone FROM Personnes WHERE age = 21

Personnes

num	nom	telephone	age
1	Rachid	0360665533	20
2	Nadia	0261003300	21
3	Karim	NULL	22
4	Ahmed	0162344343	21
5	Youssef	NULL	19

Jointure des tables :

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

En général, les jointures consistent à associer des lignes de 2 tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table.

Jointure des tables :

Produit catésien :

professeurs

som	nomp	codem
1	Khalid	IF
2	Hamid	MT
3	Safaa	IF

matieres

code	nom
IF	Informatique
MT	Mathématique

Jointure des tables :

Produit catésien :

SELECT * FROM professeurs, matieres

som	nomp	codem	code	nom
1	Khalid	IF	IF	Informatique
1	Khalid	IF	MT	Mathématique
2	Hamid	MT	IF	Informatique
2	Hamid	MT	MT	Mathématique
3	Safaa	IF	IF	Informatique
3	Safaa	IF	MT	Mathématique

Jointure des tables :

Produit catésien avec une sélection :

```
SELECT * FROM professeurs, matieres WHERE codem=code
```

som	nomp	codem	code	nom
1	Khalid	IF	IF	Informatique
2	Hamid	MT	MT	Mathématique
3	Safaa	IF	IF	Informatique

Jointure des tables :

En SQL première syntaxe :

```
SELECT * FROM professeurs, matieres  
WHERE codem=code
```

En SQL deuxième syntaxe :

```
SELECT * FROM professeurs JOIN matieres  
ON codem=code
```

Trier les données :

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule.

Trier les données :

Eleves

Num	Nom	DateNaissance
1	Aziz	11-01-1991
2	Malak	02-04-1990
3	Aziz	10-01-1990
4	Ihssan	23-03-1990

Tri croissant par Nom:

```
SELECT * FROM Eleves ORDER BY Nom
```

Num	Nom	DateNaissance
1	Aziz	11-01-1991
3	Aziz	10-01-1990
4	Ihssan	23-03-1990
2	Malak	02-04-1990

Trier les données :

Eleves

Num	Nom	DateNaissance
1	Aziz	11-01-1991
2	Malak	02-04-1990
3	Aziz	10-01-1990
4	Ihssan	23-03-1990

Tri croissant par Date de Naissance:

`SELECT * FROM Eleves ORDER BY DateNaissance`

Num	Nom	DateNaissance
3	Aziz	10-01-1990
4	Ihssan	23-03-1990
2	Malak	02-04-1990
1	Aziz	11-01-1991

Trier les données :

Eleves

Num	Nom	DateNaissance
1	Aziz	11-01-1991
2	Malak	02-04-1990
3	Aziz	10-01-1990
4	Ihssan	23-03-1990

Tri décroissant par Nom:

```
SELECT * FROM Eleves ORDER BY Nom Desc
```

Num	Nom	DateNaissance
2	Malak	02-04-1990
4	Ihssan	23-03-1990
1	Aziz	11-01-1991
3	Aziz	10-01-1990

Exercice 1 :

Soit la base de données "football" avec une table joueurs :

id	nom	poste	age	equipe	but
1	Hakimi	Défenseur	25	PSG	3
2	En-Nesyri	Attaquant	27	Séville	15
3	Amrabat	Milieu	28	Manchester Utd	2
4	Boufal	Ailier	30	Al-Rayyan	5
5	Aboukhlaïl	Attaquant	24	Toulouse	9

Exercice 1 :

Afficher uniquement les noms et les équipes des joueurs.

```
SELECT nom, equipe FROM joueurs;
```

Afficher les noms et buts marqués.

```
SELECT nom, buts FROM joueurs;
```

Afficher toutes les colonnes sauf id.

```
SELECT nom, poste, age, equipe, buts FROM joueurs;
```

Afficher les joueurs dont le poste est "Attaquant".

```
SELECT * FROM joueurs WHERE poste = 'Attaquant';
```

Afficher les joueurs de l'équipe PSG.

```
SELECT * FROM joueurs WHERE equipe = 'PSG';
```

Afficher les joueurs qui ont marqué plus de 5 buts.

```
SELECT * FROM joueurs WHERE buts > 5;
```

Exercice 1 :

Afficher les joueurs âgés de moins de 27 ans.

```
SELECT * FROM joueurs WHERE age < 27;
```

Afficher le nom et le nombre de buts des attaquants seulement.

```
SELECT nom, buts FROM joueurs WHERE poste = 'Attaquant';
```

Afficher les noms des joueurs du Toulouse ayant marqué au moins 5 buts.

```
SELECT nom FROM joueurs WHERE equipe = 'Toulouse' AND buts >= 5;
```

Afficher les noms et âges des joueurs non attaquants.

```
SELECT nom, age FROM joueurs WHERE poste <> 'Attaquant';
```

Afficher tous les joueurs classés par nombre de buts décroissant.

```
SELECT * FROM joueurs ORDER BY buts DESC;
```

Exercice 2 :

Soit la base de données "football" avec une table **matches** :

id	equipe_dom	equipe_ext	buts_dom	buts_ext	stade	date_match
1	PSG	Marseille	3	1	Parc des Princes	2024-09-15
2	Real Madrid	Barça	2	2	Bernabéu	2024-10-02
3	Liverpool	Man City	1	3	Anfield	2024-10-10
4	Bayern	Dortmund	4	2	Allianz Arena	2024-09-29
5	PSG	Monaco	2	0	Parc des Princes	2024-10-20
6	Milan	Inter	1	1	San Siro	2024-09-12

Exercice 2 :

1. Afficher la liste de toutes les équipes à domicile et à l'extérieur

```
SELECT equipe_dom, equipe_ext FROM matchs;
```

2. Afficher le nom du stade et la date de chaque match

```
SELECT stade, date_match FROM matchs;
```

3. Afficher les matchs joués au Parc des Princes

```
SELECT * FROM matchs
```

```
WHERE stade = 'Parc des Princes';
```

4. Afficher les matchs où le PSG a joué à domicile

```
SELECT * FROM matchs
```

```
WHERE equipe_dom = 'PSG';
```

Exercice 2 :

5. Afficher les matchs où l'équipe à domicile a marqué plus de 2 buts

```
SELECT * FROM matches  
WHERE buts_dom > 2;
```

6. Afficher les matchs terminés sur un score nul (égalité)

```
SELECT * FROM matches  
WHERE buts_dom = buts_ext;
```

7. Afficher les matchs joués avant le 1er octobre 2024

```
SELECT * FROM matches  
WHERE date_match < '2024-10-01';
```

8. Afficher les matchs où l'équipe à l'extérieur a gagné

```
SELECT * FROM matches  
WHERE buts_ext > buts_dom;
```

Exercice 2 :

9. Afficher les noms et scores des matchs disputés au Bernabéu

```
SELECT equipe_dom, equipe_ext, buts_dom, buts_ext  
FROM matchs WHERE stade = 'Bernabéu';
```

10. Afficher les matchs où au moins une des équipes a marqué 3 buts ou plus

```
SELECT * FROM matchs  
WHERE buts_dom >= 3 OR buts_ext >= 3;
```

11. Afficher les matchs où le PSG est impliqué (domicile ou extérieur)

```
SELECT * FROM matchs  
WHERE equipe_dom = 'PSG' OR equipe_ext = 'PSG';
```

12. Afficher le nom du stade et la date des matchs où l'équipe à domicile a marqué exactement 2 buts

```
SELECT stade, date_match FROM matchs  
WHERE buts_dom = 2;
```

Exercice 2 :

13. Afficher tous les matchs du mois d'octobre 2024

```
SELECT * FROM matchs  
WHERE date_match BETWEEN '2024-10-01' AND '2024-10-31';
```

14. Afficher les matchs dont le stade commence par la lettre "A"

```
SELECT * FROM matchs  
WHERE stade LIKE 'A%';
```

15. Afficher les matchs où le nombre total de buts (dom + ext) est supérieur à 4

```
SELECT * FROM matchs  
WHERE (buts_dom + buts_ext) > 4;
```

Exercice 3 :

Soit la base da données suivante:

livres

id_livre	titre	auteur	annee	cat_id	prix
1	L'Étranger	Albert Camus	1942	1	90
2	Le Petit Prince	Antoine de Saint-Exupéry	1943	1	75
3	Les Misérables	Victor Hugo	1862	1	120
4	Harry Potter	J.K. Rowling	1998	2	110
5	Le Seigneur des Anneaux	J.R.R. Tolkien	1954	2	150
6	Sapiens	Yuval Noah Harari	2011	3	130

categories

id_categorie	nom_categorie
1	Littérature classique
2	Fantastique
3	Essai
4	Science-fiction

Exercice 3 :

1. Afficher le titre et l'auteur de tous les livres

```
SELECT titre, auteur FROM livres
```

2. Afficher uniquement les livres publiés après 1950

```
SELECT * FROM livres
```

```
WHERE annee > 1950
```

3. Afficher les livres dont le prix est supérieur à 100

```
SELECT * FROM livres
```

```
WHERE prix > 100
```

4. Afficher les titres des livres écrits par Victor Hugo ou Albert Camus

```
SELECT titre FROM livres
```

```
WHERE auteur IN ('Victor Hugo', 'Albert Camus')
```

Exercice 3 :

5. Afficher tous les livres dont la catégorie_id vaut 2

```
SELECT * FROM livres  
WHERE cat_id = 2
```

6. Afficher le titre et le prix des livres de la catégorie 1

```
SELECT titre, prix FROM livres  
WHERE cat_id = 1
```

7. Afficher les livres dont le titre contient le mot "Le"

```
SELECT * FROM livres  
WHERE titre LIKE '%Le%'
```

8. Afficher les livres triés par prix décroissant

```
SELECT * FROM livres  
ORDER BY prix DESC
```

Exercice 3 :

9. Afficher le titre et l'année de publication des livres par ordre chronologique

```
SELECT titre, annee FROM livres ORDER BY annee ASC;
```

10. Afficher le titre du livre et le nom de sa catégorie

```
SELECT titre, nom_categorie FROM livres  
JOIN categories ON cat_id = id_categorie;
```

11. Afficher le titre, l'auteur et le nom de la catégorie de chaque livre

```
SELECT titre, auteur, nom_categorie FROM livres  
JOIN categories ON cat_id = id_categorie;
```

12. Afficher uniquement les livres appartenant à la catégorie "Fantastique"

```
SELECT * FROM livres  
JOIN categories c ON cat_id = id_categorie  
WHERE nom_categorie = 'Fantastique';
```

Expressions de calcul :

- En SQL on peut effectuer des opérations de calcul en ligne.
- On peut appliquer ces opération dans la projection ou la sélection.

Expressions de calcul :

Table : ventes

client	produit	prix	quantité
Ali	PC	3000	2
Ahmed	Clavier	50	10
Rachid	Souris	20	50

Donner le total de chaque vente ?

client	produit	total
Ali	PC	6000
Ahmed	Clavier	500
Rachid	Souris	1000

SELECT client, produit, prix*quantité as total **FROM** ventes

Expressions de calcul :

Table : ventes

client	produit	prix	quantité
Ali	PC	3000	2
Ahmed	Clavier	50	10
Rachid	Souris	20	50

Donner la liste des ventes avec un total ≥ 1000 ?

client	produit	prix	quantité
Ali	PC	3000	2
Rachid	Souris	20	50

SELECT * FROM ventes WHERE prix*quantite $\geq$ 1000

Fonctions agrégatives (Calcul simple)

- Elles permettent d'effectuer des calculs verticaux pour l'ensemble ou un sous-ensemble des valeurs d'une colonne.
- Les fonctions principales sont les suivantes :

FONCTIONS	DESCRIPTION
SUM	Permet d'effectuer la somme des valeurs d'une colonne numérique
AVG	Permet d'effectuer la moyenne des valeurs d'une colonne numérique
MAX	Permet de rechercher la valeur maximale d'une colonne numérique
MIN	Permet de rechercher la valeur minimale d'une colonne numérique
COUNT	Permet de compter le nombre de valeurs d'une colonne

Fonctions agrégatives (Calcul simple)

Exemples :

Nombre des élèves :

```
SELECT count(*) FROM eleves
```

Max des âges des élèves:

```
SELECT MAX(age) FROM eleves
```

Nombre des élèves âgés plus que 21 ans :

```
SELECT COUNT(*) FROM eleves WHERE age >=21
```

Moyenne des âges :

```
SELECT AVG(age) FROM eleves
```

eleves

num	nom	telephone	age
1	Rachid	0360665533	20
2	Nadia	0261003300	21
3	Karim	NULL	22
4	Ahmed	0162344343	21
5	Youssef	NULL	19

Les opérateurs ensemblistes

Union :

```
SELECT atr1, atr2 FROM table1  
UNION  
SELECT atr1, atr2 FROM table2
```

Intersection :

```
SELECT atr1, atr2 FROM table1  
INTERSECT  
SELECT atr1, atr2 FROM table2
```

Différence :

```
SELECT atr1, atr2 FROM table1  
MINUS  
SELECT atr1, atr2 FROM table2
```

Les opérateurs ensemblistes

Exemple :

Liste des noms des fonctionnaires
et des employés :

```
SELECT Nom FROM fonctionnaires
```

```
UNION
```

```
SELECT Nom FROM employes
```

fonctionnaires

Num	Nom	Prenom
1	Kourchi	Khalid
2	Labyad	Nabil
3	Lahmoudi	Nabila
4	Rochdi	Ahmed

employes

Num	Nom	Prenom
1	Kourchi	Khalid
2	Labyad	Nabil
3	Lahmoudi	Nabila
4	Rochdi	Ahmed

Exercice 4 :

Table : employes

id_emp	nom	poste	salaire	prime	dept	age
1	Ali	Ingénieur	8000	1000	IT	30
2	Sara	Analyste	7000	800	IT	28
3	Karim	Comptable	6000	500	FIN	40
4	Leila	Directeur	12000	2000	FIN	45
5	Youssef	Technicien	5000	400	IT	35

Exercice 4 :

1. Donner la liste des noms des salarié avec la rémunération totale de chaque employé

```
SELECT nom, (salaire+prime as) renum_total FROM employes
```

2. Afficher la liste des noms des salariés avec leur revenu annuel

```
SELECT nom, (salaire+prime)*12 as revenu FROM employes
```

3. Donner le salaire maximal des employées

```
SELECT MAX(salaire) FROM employées
```

4. Donner le salaire minimal des employées

```
SELECT MIN(salaire) FROM employées
```

5. Donner le salaire moyen des employées

```
SELECT AVG(salaire) FROM employées
```

6. Donner la somme des salaires et primes des employées

```
SELECT SUM(salaire+prime) FROM employées
```

Le partitionnement

Il doit permettre d'effectuer des calculs statistiques pour chaque sous-ensemble de lignes vérifiant un même critère. Le partitionnement s'exprime en SQL par la commande GROUP BY suivie du nom des colonnes de partitionnement.

```
SELECT < liste des colonnes de la table résultat>  
FROM < liste des tables impliquées dans l'interrogation>  
[ WHERE < condition de sélection des lignes>]  
[ GROUP BY < attributs de regroupement>]  
[ HAVING < condition sur le regroupement>]  
[ ORDER BY <attribut de tri>]
```

Le partitionnement:

- Exemple 1 :

ventes			
client	produit	prix	quantité
Ali	PC	3000	2
Ahmed	Clavier	50	10
Rachid	Souris	20	50
Meriem	PC	3000	10
Fatima	Clavier	50	30

➤ Donner pour chaque produit le total des quantités vendue :

produit	quantité
PC	12
Clavier	40
Souris	50

Le partitionnement :

- Exemple 1 :

Etape 1 : Le regroupement

client	produit	prix	quantité
Ali	PC	3000	2
Meriem	PC	3000	10
Ahmed	Clavier	50	10
Fatima	Clavier	50	30
Rachid	Souris	20	50

Etape 2 : Le calcul de la somme

produit	sum(quantité)
PC	12
Clavier	40
Souris	50

```
SELECT produit, SUM(quantite)  
FROM ventes  
GROUP BY produit
```

Le partitionnement :

- Exemple 2 :

Etape 1 : Le regroupement

client	produit	prix	quantité
Ali	PC	3000	2
Ahmed	Clavier	50	10
Ali	Souris	20	50
Meriem	PC	3000	10
Ahmed	Clavier	50	30



Etape 2 : Le calcul de la somme

client	SUM(quantité)
Ali	52
Ahmed	20
Meriem	10

```
SELECT client, SUM(quantite)
FROM ventes
GROUP BY client
```

Le partitionnement :

La clause : **HAVING**

La clause HAVING permet d'appliquer une condition de sélection sur les regroupements

Exemple :

personnels(num, nom, fonction , salaire)

Sélectionner les fonctions dont le salaire moyenne est supérieur à 1500 :

```
SELECT fonction, AVG(salaire) AS moy  
FROM personnels  
GROUP BY fonction  
HAVING moy >1500
```

Les requêtes imbriquées (sous-requête)

Définition :

- Une sous-requête est une requête à **l'intérieur** d'une autre requête.
- Avec le SQL, vous pouvez construire des requêtes imbriquées sur autant de niveaux que vous voulez.

Raisons motivant l'utilisation de sous-requêtes

- Elle permettent de décomposer une requête complexe en une série d'étapes logiques
- Elles permettent de répondre à une requête qui repose sur les résultats d'une autre requête

Les requêtes imbriquées (sous-requête)

Trois types de sous-requêtes imbriquées :

- Renvoi d'une valeur unique
- Renvoi d'une liste de valeurs
- Renvoi une relation

Les requêtes imbriquées (sous-requête)

Sous requête qui renvoi une valeur unique

Exemple 1: Notes(Id, Nom, Prenom, Moyenne)

Liste des noms et prénoms des élèves qui ont eu la moyenne maximale :

En SQL :

```
SELECT Nom, Prenom FROM Notes  
WHERE Moyenne = (SELECT MAX(Moyenne) FROM Notes)
```

Les requêtes imbriquées (sous-requête)

Sous requête qui renvoi une valeur unique

Exemple 2: personnels(num, nom, prenom, fonction, salaire)

Liste des noms et prénoms des élèves qui ont eu la moyenne maximale :

En SQL :

```
SELECT * FROM personnels  
WHERE fonction = (SELECT fonction FROM personnels  
WHERE nom='Aziz' AND prenom='Yassine')
```

Les requêtes imbriquées (sous-requête)

Sous requête qui renvoi plusieurs valeurs

Exemple 1:

employes(num, nom, fonction)

commandes(num, nume*, date, montant)

Liste des commandes qui ont été traitées par des employées qui ne sont pas des agents commerciaux ?

Les requêtes imbriquées (sous-requête)

Sous requête qui renvoi plusieurs valeurs

En SQL 1:

```
SELECT * FROM employes, commandes WHERE employes.num=nume AND  
fonction!='Agent commercial'
```

En SQL 2:

```
SELECT * FROM commandes  
WHERE nume IN (SELECT num FROM employes WHERE fonction != 'Agent  
comemrcial')
```