



Université de l'Unité Africaine (UUA)

Année universitaire : 2020 – 2021

Filière: Informatique de Gestion

Niveau: 2^{ème} année de Licence (IG2)

Module : Gestion de base de données Oracle

Enseignant : Mr. SANA

Bases de données et langage SQL

Chap : Structured Query Language (SQL)

Enseignant : Mr. SANA

Table des matières

I.	INTRODUCTION A SQL	4
II.	DEFINITION DE SCHEMA D'UNE BD	5
1.	CREATION DE TABLES.....	5
a.	<i>Syntaxe :.....</i>	5
b.	<i>Contraintes de relation.....</i>	5
c.	<i>Contraintes de colonnes.....</i>	5
d.	<i>Exemples.....</i>	5
2.	SUPPRESSION D'UNE TABLE.....	6
e.	<i>Syntaxe</i>	6
f.	<i>Exemple.....</i>	6
3.	MODIFICATION D'UN SCHÉMA.....	6
a.	<i>Syntaxe</i>	6
III.	GESTION DES DROITS ET DES PRIVILEGES	6
1.	CONTROLE DES DROITS D'ACCES.....	7
2.	HIERARCHIE DES PRIVILEGES.....	7
3.	DEFINITION DES DROITS	7
a.	<i>Syntaxe - GRANT</i>	7
b.	<i>Exemple - GRANT.....</i>	7
c.	<i>Syntaxe - REVOKE</i>	8
d.	<i>Exemple - REVOKE</i>	8
4.	REGLES D'OCTROI ET DE SUPPRESSION DES DROITS.....	8
IV.	LES ORDRES LMD	8
1.	INSERT.....	8
a.	<i>Objectif et syntaxe.....</i>	8
b.	<i>Exemples.....</i>	9
2.	UPDATE.....	9
a.	<i>Objectif et syntaxe.....</i>	9
b.	<i>Exemples.....</i>	9
3.	DELETE	9
a.	<i>Objectif et syntaxe.....</i>	9
b.	<i>Exemples.....</i>	10
V.	INTERROGATION DE DONNEES SOUS SQL.....	10
1.	REQUETES SIMPLES ET PREDICATS.....	10
a.	<i>Expression des projections.....</i>	10
b.	<i>Restriction sur les données</i>	11
c.	<i>Expression des jointures</i>	14
d.	<i>Tri des lignes</i>	15
2.	REQUETES IMBRIQUEES – SOUS REQUETES	16
a.	<i>Les types.....</i>	16
b.	<i>L'opérateur logique ALL.....</i>	17
c.	<i>L'opérateur logique ANY.....</i>	17
d.	<i>L'opérateur logique EXISTS</i>	18
3.	REQUETES COMPOSEES	19
a.	<i>L'opérateur UNION.....</i>	20
b.	<i>L'opérateur UNION ALL.....</i>	20
c.	<i>L'opérateur INTERSECT</i>	20
d.	<i>L'opérateur MINUS</i>	20
4.	LES FONCTIONS DE MULTI-LIGNES – FONCTIONS D'AGREGATION.....	21

<i>a.</i>	<i>Création de groupe : GROUP BY.....</i>	21
<i>b.</i>	<i>Exclure des groupes : HAVING.....</i>	22

I. Introduction à SQL

SQL (sigle de Structured Query Language, en français langage de requête structurée) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. SQL est devenu le langage standard pour décrire et manipuler les BDR.

- **Langage de Définition de Données (LDD)**, en anglais **Data Definition Language (DDL)** : permet de créer des bases de données, des tables, des index, des contraintes, etc.
Exemple : CREATE, ALTER, DROP, ...
- **Langage de Manipulation de Données (LMD)**, en anglais **Data Manipulation Language (DML)** : permet de manipuler les données.
Exemple : INSERT, UPDATE, DELETE, ...
- **Langage d'Interrogation de Données (LID)**, en anglais **Data Query Language (DQL)** : permet d'extraire et interroger des données.
Exemple : SELECT, ...
- **Langage de Contrôle de Données (LCD)**, en anglais **Data Control Language (DCL)** : permet de gérer les droits d'accès aux tables.
Exemple : GRANT, REVOKE, ...
- **Langage de Contrôle de Transaction (LCT)**, en anglais **Transaction Control Language (TCL)** : permet de contrôler la bonne exécution des transactions.
Exemple : COMMIT, ROLLBACK, ...
- **SQL Intégré** en anglais **Embedded SQL** : Eléments procéduraux que l'on a intégré à un langage hôte.
Exemple : SET, DECLARE CURSOR, OPEN, FETCH ...

NB : Dans toute la suite de ce cours, le symbole « | » signifie « ou ».

II. Définition de schéma d'une BD

- Création d'un schéma (Base de données, tables, index, contraintes, etc.)
Syntaxe générale : CREATE SCHEMA AUTHORISATION <nom_schema>

1. Crédation de tables

a. Syntaxe :

```
CREATE TABLE <nom_table>
(<colonne1> type1 [NOT NULL|NULL], [CONTRAINTE DE COLONNE1]
<colonne2> type2 [NOT NULL|NULL], [CONTRAINTE DE COLONNE2]
<colonne3> type3 [NOT NULL|NULL], [CONTRAINTE DE COLONNE3]
```

b. Contraintes de relation

[PRIMARY KEY (liste de colonnes)], [UNIQUE (liste de colonnes)], etc.
FOREIGN KEY (liste de colonnes) REFERENCES table

[ON UPDATE {RESTRICT | CASCADE | SET NULL}], etc.
[CHECK condition]);

c. Contraintes de colonnes

- Les contraintes de colonnes permettent de spécifier différentes contraintes d'intégrité portant sur un seul attribut, y compris les contraintes référentielles.
- Les différentes variantes possibles sont :
 - Valeur nulle impossible (syntaxe **NOT NULL**),
 - Unicité de l'attribut (syntaxe **UNIQUE** ou **PRIMARY KEY**),
 - Contraintes de référence : syntaxe **REFERENCES** <table référencée> [(colonne référencée>)], le nom de la colonne référencée étant optionnel s'il est identique à celui de la colonne référençante,
 - Contrainte générale (syntaxe **CHECK** <condition>) ; la condition est une condition pouvant spécifier des plages ou des listes de valeurs possibles.
- Les contraintes de relations peuvent porter sur plusieurs attributs.
- Ce peut être des contraintes d'unicité, référentielles ou générales.
- Elles sont exprimées à l'aide des phases suivantes :
 - contraintes d'unicité **UNIQUE** <attribut>+,
 - contraintes référentielles permettant de spécifier quelles colonnes référencent celles d'une autre table, [**FOREIGN KEY** (<colonne référençante>+)] **REFERENCES** <table référencée> [(colonnes référencées>+)]

d. Exemples

Création des tables « Pilote », « Avion », « Vol »

```
Pilote (numpil, nompil, adr, sal)
Avion (numav, nomav, cap, loc)
Vol (numvol, #numpil, #numav, villedep, villearr, datedep, datearr, hdep, harr)
```

```
CREATE TABLE Pilote (numpil INT NOT NULL, nompil CHAR(20) NOT  
NULL, adr CHAR(50) NOT NULL, sal DEC (8, 3) DEFAULT 0, PRIMARY  
KEY(numpil));
```

```
CREATE TABLE Avion (numav INT NOT NULL, nomav CHAR(20) NOT  
NULL, cap INT NOT NULL, loc CHAR(20), PRIMARY KEY(numav));
```

```
CREATE TABLE Vol (numvol INT NOT NULL, numpil INT NOT NULL,  
numav INT NOT NULL, villedep CHAR(30) NOT NULL, villearr  
CHAR(20) NOT NULL, hdep DEC(2,2), harr DEC(2,2),  
PRIMARY KEY(numvol)  
FOREIGN KEY numpil REFERENCES Pilote,  
FOREIGN KEY numav REFERENCES Avion,  
CHECK (hdep BETWEEN 1 AND 24),  
CHECK (harr BETWEEN 1 AND 24));
```

2. Suppression d'une table

e. Syntaxe

```
DROP TABLE <nom_table>;
```

f. Exemple

```
DROP TABLE Avion;
```

3. Modification d'un schéma

a. Syntaxe

```
ALTER TABLE <nom_de_la_table> <Altération> ;
```

Différents types d'altérations sont possibles :

- Ajout d'une colonne (**ADD COLUMN**)
- Modification de la définition d'une colonne (**ALTER COLUMN**)
- Suppression d'une colonne (**DROP COLUMN**)
- Ajout d'une contrainte (**ADD CONSTRAINT**)
- Suppression d'une contrainte (**DROP CONSTRAINT**)

Exemple:

```
ALTER TABLE Pilote ADD COLUMN `Nationalite` CHAR(30) NOT NULL DEFAULT 'BF';
```

III. Gestion des droits et des privilèges

1. Contrôle des droits d'accès

Objectif: Protéger les données de la base contre les accès non autorisés

Deux niveaux:

- Connexion au serveur restreinte aux usagers répertoriés (mot de passe)
- Privilège d'accès aux objets de la base

Granule d'objet:

Vue, relation, procédure stockée, Trigger

2. Hiérarchie des privilèges

Administrateur système {users, login, ...}



Administrateur de la base {connexion utilisateurs à BD, gestion des schémas}



Propriétaires d'Objets {créateurs d'objets, distributions de droits}



Autres (Public) {droits obtenus par GRANT}

- La gestion des droits d'accès aux tables est décentralisée : il n'existe pas d'administrateur global attribuant des droits.
- Chaque créateur de table obtient tous les droits d'accès à cette table, en particulier les droits d'effectuer les actions de :
 - Sélection (**SELECT**),
 - D'insertion (**INSERT**),
 - De Suppression (**DELETE**),
 - De mis à jour (**UPDATE**),
 - Et aussi de référencer la table dans une contrainte (**REFERENCES**)
- Il peut ensuite passer ses droits sélectivement à d'autres utilisateurs ou à tout le monde (**PUBLIC**).
- Un droit peut être passé avec le droit de le transmettre (**WITH GRANT OPTION**) ou non.

3. Définition des droits

a. Syntaxe – GRANT

GRANT <PRIVILEGES> **ON** [TABLE] <Nom de la table> **TO** <RECEPTEUR>+ [**WITH GRANT OPTION**]

- Avec : <PRIVILEGES> ::= ALL PRIVILEGES | <ACTION>+
- <ACTION> ::= **SELECT** | **INSERT** | **UPDATE** [(<Nom de colonne>+)] **REFERENCE** [(<Nom de colonne>+)]
- <RECEPTEUR> ::= **PUBLIC** | <IDENTIFIANT D'AUTORISATION>

b. Exemple – GRANT

- Exemple : **GRANT SELECT, UPDATE** (numpil) **ON TABLE Vol TO OUEDRAOGO WITH GRANT OPTION**

c. Syntaxe – REVOKE

REVOKE <privilège> **ON** [TABLE] <nom de table> **FROM** <récipient>

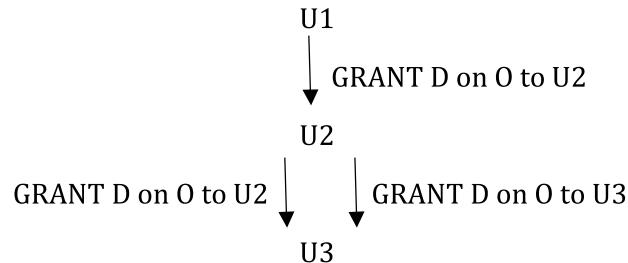
d. Exemple – REVOKE

REVOKE ALL ON TABLE Vol FROM OUEDRAOGO

=> La commande permet de retirer le droit auparavant donné ainsi que tous les droits qui en dépendent (c'est-à-dire ceux de sélection ou mise à jour de la table Vol passés par DUPONT)

4. Règles d'octroi et de suppression des droits

- On ne peut transmettre que les droits que l'on possède ou qui nous ont été transmis avec « GRANT OPTION ».
- On ne peut supprimer que les droits que l'on a transmis.
- La révocation des droits est récursive.
 - Si un utilisateur U1 a reçu le droit D de la part de plusieurs utilisateurs (U2, U3), il ne perd ce droit que si tous les utilisateurs le lui retirent
 - Ces règles garantissent la cohérence des retraits même en cas de graphe cyclique.



IV. Les ordres LMD

1. INSERT

a. Objectif et syntaxe

Objectif : Permet d'insérer des données dans une table.

Syntaxe :

INSERT INTO <Nom_de_la_table> [(<Nom de colonne> +)]
{VALUES (<CONSTANTE>+) | <Commande de recherche>} ;

Remarques :

- Dans le cas où la liste de colonnes n'est pas spécifiée, tous les attributs de la relation doivent être fournis dans l'ordre de déclaration.
- Si seulement certaines colonnes sont spécifiées, les autres sont insérées avec la valeur nulle.

b. Exemples

Exemple 1:

```
INSERT INTO Pilote (Numpil, Nompil, adr, sal)  
Values (12345, "ZONGO Richard", "Tanghin", 10000)
```

Exemple 2:

```
INSERT INTO PiloteServ  
SELECT V. Numpil, Nompil, Numvol  
FROM Pilote P, Vol V  
WHERE P.Numpil = V.Numpil;
```

Remarque : Une insertion à partir d'une commande de recherche permet de composer une relation (ici **PiloteServ**) à partir des tuples d'une relation existante (ici Pilote et vol) par recherche dans la base.

2. UPDATE

a. Objectif et syntaxe

Objectif : Permet de modifier des données dans une table.

Syntaxe :

```
UPDATE <Nom_de_la_table>  
SET {<Nom_de_colonne> = {<Expression de valeur> | NULL}} +  
WHERE {<Condition de recherche> | CURRENT OF <Nom de curseur>}
```

b. Exemples

Exemple 1:

```
UPDATE Avion SET adr = "Ouagadougou"  
WHERE Numpil = 12345;
```

Exemple 2:

```
UPDATE Avion SET cap = cap -10  
WHERE nomav LIKE "boeing 747"  
AND loc LIKE "NICE";
```

3. DELETE

a. Objectif et syntaxe

Objectif : Permet de supprimer des données dans une table.

Syntaxe :

DELETE FROM <Nom_de_la_table>
WHERE {<Condition de recherche> | CURRENT OF <Nom de curseur>}

Remarque : (ATTENTION) S'il n'y a pas de clause WHERE, toutes les lignes sont supprimées.

b. Exemples

Exemple 1: **DELETE FROM Avion**
 WHERE cap = 0;

Exemple 2: **DELETE FROM Avion**
 WHERE cap = 0
 And numav NOT IN (SELECT Distinct numav
 FROM Vol);

V. Interrogation de données sous SQL

1. Requêtes simples et prédictats

Syntaxe générale :

SELECT [ALL | DISTINCT] <expression de valeurs>+
FROM <Liste de table> [<alias>]

Projection

WHERE <Liste de critères de sélection ou prédictats>

Restriction

AND <Liste de critères de jointures>

Jointures

ORDER BY <colonne>+ [DESC | ASC]

Tri des tuples

a. Expression des projections

SQL permet d'appliquer des fonctions de calculs (+, -, * et /) sur les colonnes extraites.

Exemple 1 : Afficher les numéros des pilotes et leurs salaires multipliés par 1,1

SELECT numpil, (sal * 1,10) AS salaire_pilote_augmente FROM Pilote ;

SQL n'élimine pas les doublons, à moins que cela soit explicitement demandé par le mot clé **DISTINCT**, l'option par défaut étant **ALL**.

- A un coût, donc à ne pas utiliser lorsque c'est inutile

- Lorsque la table résultat contient un attribut (ou un ensemble d'attributs) qui pourrait être clé primaire, il est inutile d'utiliser DISTINCT
- Si l'on veut éliminer les lignes identiques de la table résultat et si cette table ne possède pas d'attribut (ou d'ensemble d'attributs) pouvant servir de clé primaire, alors l'utilisation de DISTINCT est obligatoire.

Exemple 2 :

```
SELECT DISTINCT numVoyage, nomVoyage, type Pension FROM Voyage ;
```

DISTINCT est inutile car numVoyage est une clé du résultat ;

Exemple 3 :

```
SELECT DISTINCT numVille FROM Etape  
WHERE numVoyage = 9 ;
```

DISTINCT obligatoire car numVille n'est pas une clé primaire de la table « Etape » du résultat.

Exemple 3 :

Table Vente

numéro	référence_produit	numéro_client	date
00102	153	101	12110/04
00809	589	108	20101/05
11005	158	108	15103/05
12005	5139	125	30103105

$R1 \leftarrow \pi_{\text{référence_produit}, \text{numéro_client}}(\text{Vente})$

```
SELECT référence_produit, numéro_client FROM Vente ;
```



référence_produit	numéro_client
153	101
589	108
158	108
5139	125

b. Restriction sur les données

$R1 \leftarrow \sigma_{\text{Adresse} = \text{"Tenkodogo"}}(\text{Client})$

$R2 \leftarrow \pi_{\text{nom}, \text{Adresse}}(R1)$

```
SELECT nom, adresse  
FROM Client  
WHERE adresse = "Tenkodogo" (adresse LIKE "Tenkodogo")
```

Numéro client	Nom	Adresse	Téléphone
101	ZONGO	Ouagadougou	78010203
106	SABA	Tenkodogo	70456123
110	OUEDRAOGO	Tenkodogo	76987654
125	SAWADOGO	Koudougou	79123456

i. Restriction sur les données : Prédicats de restriction

- Une condition de recherche élémentaire est appelée prédicat en SQL.
- Une condition de recherche définit un critère, qui appliqué à un tuple, est :
 - ✓ Vrai,
 - ✓ Faux,
 - ✓ ou inconnu,
 - ✓ Selon le résultat de l'application d'opérateurs booléens (AND, OR, NOT) a des conditions élémentaires.
- Un prédicat de restriction permet de comparer deux expressions de valeurs ;
 - ✓ La première valeur contenant des spécifications de colonnes est appelée terme.
 - ✓ La seconde contenant seulement des spécifications de constantes est appelée constante.
- Il existe une grande diversité de prédicats en SQL. On trouve en effet :
 - ✓ Un prédicat de **comparaison** {"=", "!=","<",">","<=",">="},
 - ✓ Un prédicat d'intervalle **BETWEEN**
 - ✓ Un prédicat de comparaison de texte **LIKE**
 - ✓ Un prédicat de test de nullité (**IS NULL / IS NOT NULL**),

- ✓ Un prédicat d'appartenance **IN**.

- ❖ AND : renvoie TRUE si les deux conditions sont vraies.
- ❖ OR : renvoie TRUE si l'une des conditions est vraie.
- ❖ NOT : renvoie TRUE si la condition qui suit est FALSE et inversement.

```
SELECT nom
FROM employés
WHERE salaire >= 700 000
AND ville LIKE "%ou%";
```

Cette requête ci-dessus permet de retourner les noms des employés dont le salaire est supérieur à 700 000 et dans le libellé de la ville il y'a les caractères "ou" ;

i. Restriction sur les données : Le prédicat **BETWEEN**

Exemple

```
SELECT nom
FROM employés
WHERE salaire BETWEEN 700000 AND 900000;
```

Cette requête ci-dessus permet de retourner les noms des employés dont le salaire est compris entre 700 000 et 900 000.

ii. Restriction sur les données : Le prédicat **LIKE**

Recherche de chaîne de caractère valide à l'aide de caractères génériques.

- ❖ %: représente une chaîne de 0 à n caractères.
- ❖ _: représente un seul caractère.

Exemple

```
SELECT nom
FROM employés
WHERE ville LIKE "B%" ;
```

Cette requête ci-dessus permet de retourner les noms des employés dont le libellé de la ville commencent par la lettre "B" ;

iii. Restriction sur les données : Le prédicat **NULL**

- La vérification de la valeur NULL est effectuée par les opérateurs **IS NULL** et **IS NOT NULL**.

- Recherche des valeurs non attribuées.

Exemple 13 : Rechercher les noms d'employés (table employés) dont la ville n'est pas attribuée

```
SELECT nom
```

```
FROM employés
```

```
WHERE ville IS NULL;
```

iv. Restriction sur les données : Le prédictat **IN**

Vérifie l'appartenance d'une données à une série de valeurs.

Exemple

```
SELECT nom
FROM employes
WHERE ville IN ("Ouahigouya", "Manga");
```

↔ SELECT nom
FROM employes
WHERE Ville = " Ouahigouya" **OR** ville = " Manga"

↔ Un NOT IN sera équivalent à
WHERE Ville != " Ouahigouya" **AND** ville != " Manga"

c. Expression des jointures

- Permettent d'afficher des données de plusieurs tables
- NB : Si on ne mentionne pas la condition de jointure : elle est équivalente à un produit cartésien.

Exemple

```
SELECT nom, libel
FROM employés, services ;
```

- ✓ La condition de jointure est exprimée dans la clause WHERE.

i. Syntaxe

```
SELECT liste_des_colonnes
FROM listes_des_tables
WHERE table1.colonne1 = table2.colonne2 ;
```

- ✓ Le nom de la table est nécessaire si les noms des colonnes sont identiques.

Exemple

```
SELECT nom, libel
FROM employés, services
WHERE employés.numService = services. numService ;
```

ii. Alias de Table

- ✓ Evite les ambiguïtés sur les noms de colonnes
- ✓ Améliore les performances
- ✓ Permet de simplifier les interrogations
- ✓ Utilisation de préfixes désignant les tables

Exemple:

```
SELECT e.nom, s.libel
FROM employés e, services s
WHERE e.numService = s. numService ;
```

iii. Expression des jointures externes

- ✓ Permettent de visualiser également les lignes qui ne correspondent pas à la condition de jointure
- ✓ On distingue la jointure externe gauche (LEFT JOIN) et la jointure externe droite (Right JOIN)

Exemple:

```
SELECT e.nom, s.libel
FROM employés e RIGHT JOIN services s
ON e.numService = s. numService ;
```

d. Tri des lignes

- Permet de trier les lignes dans un résultat d'une requête SQL.

Syntaxe:

```
SELECT liste_des_colonnes
FROM listes_des_tables
[WHERE condition]
ORDER BY liste_de_colonnes [ASC | DESC] ;
```

Par défaut les résultats sont classés par ordre croissant.

Exemple : Afficher les noms des employés ayant un salaire supérieur ou égal à 300000 par ordre décroissant du salaire

```
SELECT Numemp, nom  
FROM employés  
WHERE salaire >= 300000  
ORDER BY salaire DESC ;
```

2. Requêtes imbriquées – Sous requêtes

- ❖ Requête imbriquée dans la clause WHERE d'une requête externe.

Syntaxe :

```
SELECT liste_des_colonnes  
FROM listes_des_tables  
WHERE [Opérande] Opérateur (SELECT ...  
                          FROM ...  
                          WHERE ...) ;
```

- ❖ Opérateurs ensemblistes
 - ✓ Appartenance ensembliste : (A_1, \dots, A_n) **IN** <Sous-requête>
 - ✓ Test d'existence : **EXISTS** <Sous-requête>
 - ✓ Comparaison avec quantificateur (ANY par défaut) :

(A_1, \dots, A_n) <comp> [**ALL** | **ANY**] <Sous-requête>

a. Les types

➤ La sous-requête peut renvoyer soit

- ✓ une ligne,
- ✓ soit plusieurs
 - L'opérateur IN
 - Les opérateurs obtenus en ajoutant ANY ou ALL à la suite d'un opérateur de comparaison classique ($=, <>, >, >=, <, <=$)
 - OU bien ($=, <>, >, >=, <, <=$) suivis par des sous requêtes SELECT
- ✓ Si la sous-requête renvoie plusieurs lignes, nous avons deux cas, selon que la sous-requête est indépendante de la requête principale ou synchronisée avec elle.
- ✓ Si la sous-requête est synchronisée avec la requête principale, on parle de requête corrélée.

(Besoin de jointure entre la requête et la sous-requête)

b. L'opérateur logique ALL

- Sert à comparer une valeur à toutes les valeurs d'un ensemble renvoyées par une sous-requête.
- La comparaison est vraie si elle est vraie pour tous les éléments de l'ensemble.

Exemple :

Personne (idpers, nompers, adr, poste, salaire, date_emp, #idserv)

Service (idserv, nomserv, ville)

Chercher le salarié ayant un salaire supérieur à tous les salaires du service n°10.

```
SELECT idpers, nompers FROM personne  
WHERE salaire > ALL (SELECT DISTINCT salaire  
                      FROM personne  
                      WHERE idserv = 10);
```

→ *La comparaison se fait avec tous les éléments (un par un) renvoyés par la sous-requête*

Ceci est équivalent à :

```
SELECT idpers, nompers FROM personne  
WHERE salaire > (SELECT MAX (salaire)  
                      FROM personne  
                      WHERE idserv = 10);
```

Si on veut chercher le salarié ayant un salaire inférieur à tous les salaires du service n°10.

```
SELECT idpers, nompers FROM personne  
WHERE salaire < ALL (SELECT (ou MIN) salaire  
                      FROM personne  
                      WHERE idserv = 10);
```

c. L'opérateur logique ANY

- Sert à comparer une valeur à toute valeur applicable d'un ensemble renvoyée par une sous-requête.
- La comparaison est vraie si elle est vraie pour au moins un élément de l'ensemble.

Exemple Personne (idpers, nompers, adr, poste, salaire, date_emp, #idserv)
 Service (idsev, nomserv, ville)

→ Chercher s'il y a un(e) secrétaire ayant un salaire supérieur à un ou plusieurs ingénieurs de la société.

```
SELECT idpers, nompers FROM personne  
WHERE poste="SECRETAIRE"  
  
AND salaire > ANY (SELECT salaire  
                      FROM personne  
                      WHERE poste="INGENIEUR");
```

La condition est vraie dès qu'on rencontre une ligne vérifiant la condition.

d. L'opérateur logique EXISTS

- ◆ Permet de chercher la présence d'une ligne répondant à certains critères dans une table spécifique :
- ◆ Elle est vérifiée si une sous-requête retourne au moins une ligne.

Exemple 1 : Personne (idpers, nompers, adr, poste, salaire, date_emp, #idserv)
Service (idserv, nomserv, ville)

Donner les noms des services ayant au moins un employé embauché après le 01/01/05.

```
SELECT nomserv
FROM service S
WHERE EXISTS (SELECT *
               FROM personne P
               WHERE date_emb > "01/01/05"
               AND S.idserv = P.idserv);
```

Exemple 2 : Donner les noms des services ayant au moins un employé embauché après le 01/01/05.

Exécution et Résultat ?

Table Service

idserv	nomserv	ville
S001	Finance	Bobo
S002	Vente	Ouahigouya
S003	Commercial	Banfora

```
SELECT nomserv
FROM service S
WHERE EXISTS (SELECT *
               FROM personne P
               WHERE date_emb > "01/01/05"
               AND S.idserv = P.idserv);
```

Table personne

idpers	nompers	adr	poste	Salaire	Date_emp	idserv
100					20/09/1999	S002
101					02/06/2002	S001
102					30/04/2005	S002
103					18/02/2006	S003

Pour S001, Exists = faux Donc Finance est rejeté

Pour S002, Exists = vrai Donc Vente est retourné

Pour S003, Exists = vrai Donc Commercial est retourné

Exemple 3 :

Donner les noms des projets dont aucun employé n'appartient au service FINANCE.

- Personne (idpers, nompers, adr, poste, salaire, date_emp, #idserv)
- Service (idsev, nomserv, ville)
- Projet (idproj, titreproj, #respproj)
- Persprojet (#idproj, #idpers)

```
SELECT DISTINCT titreproj
FROM projet pj
WHERE NOT EXISTS (SELECT *
                   FROM personne p, persprojet pp
                   WHERE pj.idproj = pp.idproj
                     AND pp.idpers = p.idpers
                     AND idserv = (SELECT idserv
                                   FROM service
                                   WHERE nomserv LIKE "FINANCE "));
```

Remarque sur l'utilisation de « IN » vs « = » dans les requêtes imbriquées :

Lorsque la sous-requête peut rendre plusieurs lignes on ne peut pas utiliser « = ». C'est « IN » qui doit être utilisée.

3. Requêtes composées

- ◆ Une requête composée est l'association de plusieurs requêtes par le biais d'un opérateur de combinaison.
- ◆ Syntaxe :

```
SELECT colonne 1, colonne 2 FROM TABLE1
      WHERE conditions ...
      opérateur
      (SELECT colonne1, colonne2 FROM TABLE2
      WHERE conditions);
```
- ◆ Opérateurs : UNION, UNION ALL, INTERSECT, MINUS.

a. L'opérateur UNION

- ◆ Permet de combiner les résultats de deux instructions SELECT ou plus, sans retourner des lignes en double (Pas de doublons). Les colonnes sélectionnées des différentes requêtes doivent être de même nombre, de même type et de même ordre.

Personne (idpers, nompers, adr, poste, salaire, date_emp, #idserv)
Projet (idproj, titreproj, #respproj)
Persprojet (#idproj, #idpers)

Exemple : Donner la liste des salariés travaillant sur des projets comme étant employés ou responsables de projets.

```
SELECT * FROM personne WHERE idpers IN (SELECT DISTINCT idpers FROM persprojet)
UNION
SELECT * FROM personne WHERE idpers IN (SELECT respproj FROM projet);
```

b. L'opérateur UNION ALL

- ◆ Permet de combiner les résultats de deux instructions SELECT y compris les lignes dupliquées (Retourne les doublons aussi). Les règles qui s'appliquent à cet opérateur sont les mêmes que celles qui concernent UNION.

c. L'opérateur INTERSECT

- ◆ Permet de combiner les résultats de deux instructions SELECT ou plus, mais ne retourne que les lignes en commun.

Exemple : Donner la liste des salariés qui sont à la fois affectés à des projets et responsables de projets.

```
SELECT * FROM personne WHERE idpers IN (SELECT DISTINCT idpers
                                         FROM persprojet)
```

INTERSECT

```
SELECT * FROM personne WHERE idpers IN (SELECT respproj
                                         FROM projet);
```

d. L'opérateur MINUS

- ◆ Permet de retourner les lignes d'une première instruction SELECT qui ne sont pas retournées par la deuxième instruction SELECT. Les colonnes sélectionnées des différentes requêtes doivent être de même nombre, de même type et de même ordre.
- ◆ Exemple : Donner la liste des salariés qui sont responsables de projets mais qui ne sont pas affectés à des projets comme étant employés.

```
SELECT * FROM personne WHERE idpers IN (SELECT respproj FROM projet)
```

MINUS

```
SELECT * FROM personne WHERE idpers IN (SELECT DISTINCT idpers FROM persProjet);
```

4. Les fonctions de multi-lignes – Fonctions d'agrégation

Les fonctions d'agrégation dans le langage SQL permettent d'effectuer des opérations statistiques sur un ensemble d'enregistrement. Étant données que ces fonctions s'appliquent à plusieurs lignes en même temps, elles permettent des opérations qui servent à récupérer l'enregistrement le plus petit, le plus grand ou bien encore de déterminer la valeur moyenne sur plusieurs enregistrement.

Les principales fonctions sont les suivantes :

- **AVG()** pour calculer la moyenne sur un ensemble d'enregistrement
- **COUNT()** pour compter le nombre d'enregistrement sur une table ou une colonne distincte
- **MAX()** pour récupérer la valeur maximum d'une colonne sur un ensemble de ligne. Cela s'applique à la fois pour des données numériques ou alphanumérique
- **MIN()** pour récupérer la valeur minimum de la même manière que MAX()
- **SUM()** pour calculer la somme sur un ensemble d'enregistrement

Exemple 1: SELECT **COUNT** (numav) FROM Avion ;

Exemple 2:

SELECT **COUNT (DISTINCT numav)** FROM Vol; (numav n'est pas la clé primaire de la table Vol)

Exemple 3: SELECT **SUM** (sal) FROM Pilote WHERE sal>750000;

Exemple 4: SELECT **AVG** (sal) FROM Pilote;

Exemple 5: SELECT **MIN** (sal) FROM Pilote;

Exemple 6: SELECT **MAX** (sal) FROM Pilote ;

- Les enregistrements seront regroupés grâce à la clause **GROUP BY**.
- Les enregistrements pourront être exclus à l'aide de la clause **HAVING**.

Les fonctions de groupe donneront des résultats par groupe.

a. Création de groupe : GROUP BY

Syntaxe :

```
SELECT [ ALL | DISTINCT ] {<expression de valeurs>+ | *}
FROM <nom de table>
[WHERE <condition de recherche>]
GROUP BY <SPECIFICATION DE COLONNE>+ ;
```

Exemple :

- Personne (idpers, nompers, adr, poste, salaire, date_emp, #idserv)
- Service (idsev, nomserv, ville)

```
SELECT idserv, COUNT(idpers)
FROM personne
GROUP BY idserv;
```

b. Exclude des groupes : HAVING

Syntaxe :

```
SELECT [ ALL | DISTINCT ] {<expression de valeurs>+ | *}
FROM <nom de table>
[WHERE <condition de recherche>]
GROUP BY <SPECIFICATION DE COLONNE>+
HAVING <SPECIFICATION DE VALEURS>+ ;
```

Exemple 1: Chercher les services dont le nombre d'employés est supérieur à 10.

- Personne (idpers, nompers, adr, poste, salaire, date_emp, #idserv)
- Service (idsev, nomserv, ville)

```
SELECT idserv
FROM personne
GROUP BY idserv
HAVING COUNT(idpers) > 10;
```

Exemple 2: Lister les salaires totaux par service. En gardant ceux dont le salaire moyen dépasse 300000.

```
SELECT idserv, SUM(salaire)
FROM personne p, service s
WHERE p.idserv = s.idserv
GROUP BY idserv
HAVING AVG(idpers) > 300000;
```

Exemple 3: Lister, par service, les salaires totaux et le salaire moyen tel que le salaire minimum dépasse 200000.

```
SELECT idserv, SUM(salaire), AVG(salaire)
FROM personne p, service s
WHERE p.idserv = s.idserv
GROUP BY idserv
HAVING MIN(idpers) > 200000;
```

Remarque : Pour spécifier le **nombre d'enregistrement à retourner** ou un **pourcentage** sur Access ou sur Microsoft SQL Server, on utilise la syntaxe suivante :

```
SELECT TOP number | percent nom_de_colonnes  
FROM nom_de_table  
WHERE condition;
```

Exemple : Donner les 03 personnes qui ont les salaires les plus élevés.

```
SELECT TOP 3  
FROM personne  
ORDER BY sal DESC ;
```