



**ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR**
Membre de **HONORIS UNITED UNIVERSITIES**

École Marocaine des Sciences de l'Ingénieur

Filière: IIR

Les Bases de Données

Prof. Zakaria KHATAR

LANGAGE SQL :

Langage d'interrogation des données (LID)

Langage d'interrogation des données : LID

Ce langage permet de rechercher des informations utiles en interrogeant la base de données, à travers des projections, restrictions, composé des requêtes suivantes :

Voici à quoi ressemble une requête d'interrogation d'une base de données

```
SELECT Titre FROM Film  
WHERE Acteur = 'Tom Hanks';
```

1) La clause **SELECT**

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

L'utilisation basique de cette commande s'effectue de la manière suivante :

```
SELECT nom_colonne  
FROM nom_table
```

Cette requête (**SELECT**) va sélectionner la colonne << *nom_colonne* >> provenant (**FROM**) de la table appelée << *nom_table* >>.

LANGAGE D'INTERROGATION DES DONNÉES : LID

Exemple :

Imaginons une base de données appelée « client » qui contient des informations sur les clients d'une entreprise.

Table « client » :

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

Si l'on veut avoir la liste de toutes les villes des clients, il suffit d'effectuer la requête suivante :

```
SELECT ville  
FROM client
```

LANGAGE D'INTERROGATION DES DONNÉES : LID

Résultat :

ville
Paris
Nantes
Lyon
Marseille
Grenoble

LANGAGE D'INTERROGATION DES DONNÉES : LID

Obtenir plusieurs colonnes :

Avec la même table client il est possible de lire plusieurs colonnes (attributs) à la fois. Il suffit tout simplement de séparer les noms des champs souhaités par une virgule.

Pour obtenir les prénoms et les noms des clients il faut alors faire la requête suivante:

```
SELECT prenom, nom  
FROM client
```

Résultat :

prenom	nom
Pierre	Dupond
Sabrina	Durand
Julien	Martin
David	Bernard
Marie	Leroy

Obtenir toutes les colonnes d'un tableau :

Il est possible de retourner automatiquement toutes les colonnes d'un tableau sans avoir à connaître le nom de toutes les colonnes. Au lieu de lister toutes les colonnes, il faut simplement utiliser le caractère << * >> (étoile). Il s'utilise de la manière suivante :

```
SELECT *  
FROM client
```

Résultat :

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

2) La commande DISTINCT

L'utilisation de la commande SELECT en SQL peut potentiellement afficher des lignes **en doubles**.

Pour éviter des redondances dans les résultats il faut simplement ajouter **DISTINCT** après le mot SELECT.

```
SELECT DISTINCT nom_colonne FROM nom_table
```

3) La commande AS (alias)

Dans le langage SQL il est possible d'utiliser des **alias** pour *renommer temporairement* une colonne ou une table dans une requête.

Alias sur une colonne

La syntaxe pour renommer les colonnes : 'nom_colonne' à 'Nom de famille' et 'prenom_colonne' à 'Prénom' est la suivante :

```
SELECT nom_colonne AS Nom_de_famille, prenom_colonne AS Prénom  
FROM nom_table
```

Cette syntaxe peut également s'afficher de la façon suivante :

```
SELECT nom_colonne Nom_de_famille, prenom_colonne Prénom  
FROM nom_table
```

Exemple

```
SELECT colonne1 AS Nom, colonne2 AS Prénom  
FROM etudiant
```

CNE	colonne1	colonne2
2022150	Raji	Anas
2022151	Amrani	Chaimaa
2022152	Badrane	Mounir



Nom	Prénom
Raji	Anas
Amrani	Chaimaa
Badrane	Mounir

Alias sur une table

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

```
SELECT *  
FROM nom_table AS Etudiant
```

Cette requête peut également s'écrire de la façon suivante :

```
SELECT *  
FROM nom_table Etudiant
```


4) La commande WHERE

La commande WHERE dans une requête SQL permet *d'extraire les lignes* d'une base de données qui *respectent une condition*. Cela permet d'obtenir uniquement les informations désirées.

La commande WHERE s'utilise *en complément* à une requête utilisant SELECT. La façon la plus simple de l'utiliser est la suivante :

```
SELECT nom_colonnes FROM nom_table  
WHERE condition
```

WHERE: Permet de sélectionner des lignes selon une condition donnée



prenom	nom	ville
Pierre	Dupond	Paris
Sabrina	Durand	Nantes
Julien	Martin	Lyon
David	Bernard	Marseille
Marie	Leroy	Grenoble

Exemple

Imaginons une base de données appelée << client >> qui contient le nom des clients, le nombre de commandes qu'ils ont effectuées et leur ville :

id	nom	nbr_commande	ville
1	Paul	3	paris
2	Maurice	0	rennes
3	Joséphine	1	toulouse
4	Gérard	7	paris

Pour obtenir seulement la liste des clients qui habitent a Paris, il faut effectuer la requête suivante :

```
SELECT * FROM client
WHERE ville = 'paris'
```

Résultat

id	nom	nbr_commande	ville
1	Paul	3	paris
4	Gérard	7	paris

5) Opérateurs de comparaisons

Il existe plusieurs opérateurs de comparaisons. La liste ci-jointe présente quelques uns des opérateurs les plus couramment utilisés.

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

5 -1) Opérateur BETWEEN

Il est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant **WHERE**.

L'utilisation de la commande **BETWEEN** s'effectue de la manière suivante :

```
SELECT nom_colonne  
FROM nom_table  
WHERE nom_colonne BETWEEN valeur1 AND valeur2
```

La requête suivante retournera toutes les lignes dont la valeur de la colonne “nom_colonne” sera comprise entre **valeur1** et **valeur2**.

Cette syntaxe peut être associée à l'opérateur **NOT** pour recherche toutes les lignes **qui ne sont pas comprise** entre **valeur1** et **valeur2**.

```
WHERE nom_colonne NOT BETWEEN valeur1 AND valeur2
```

5 -2) Opérateurs: AND & OR

Les opérateurs logiques AND et OR s'utilisent au sein de la **commande WHERE** pour combiner des conditions et filtrer les données.

L'opérateur **AND** permet de s'assurer que la **condition1** **ET** la **condition2** sont vrai :

```
SELECT nom_colonne  
FROM nom_table  
WHERE condition1 AND condition2
```

L'opérateur **OR** vérifie quant à lui que la **condition1** **OU** la **condition2** est vrai :

```
SELECT nom_colonne  
FROM nom_table  
WHERE condition1 OR condition2
```

LANGAGE D'INTERROGATION DES DONNÉES : LID

Ces opérateurs peuvent être combinés et mélangés. L'exemple ci-dessous filtre les résultats de la table “nom_table” si condition1 **ET** condition2 **OU** condition3 est vrai :

```
SELECT nom_colonne  
FROM nom_table  
WHERE condition1 AND (condition2 OR condition3)
```

LANGAGE D'INTERROGATION DES DONNÉES : LID

Exemple

Pour illustrer les prochaines commandes, on va considérer la table “produit” suivante :

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	32	35
3	souris	informatique	16	30
4	crayon	fourniture	147	2

❖ **Opérateur AND** : Pour filtrer uniquement les produits informatique qui sont presque en rupture de stock (**moins de 20** produits disponible) il faut exécuter la requête suivante :

```
SELECT * FROM produit
WHERE categorie = 'informatique' AND stock < 20
```

LANGAGE D'INTERROGATION DES DONNÉES : LID

Cette requête retourne les résultats suivants :

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
3	souris	informatique	16	30

❖ **Opérateur OR** : Pour filtrer les données pour avoir uniquement les données sur les produits “ordinateur” ou “clavier” il faut effectuer la recherche suivante :

```
SELECT * FROM produit  
WHERE nom = 'ordinateur' OR nom = 'clavier'
```

Cette requête retourne les résultats suivants :

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	32	35

LANGAGE D'INTERROGATION DES DONNÉES : LID

❖ Combinaison AND et OR : les opérateurs peuvent **AND** et **OR** être combinés pour effectuer de puissantes recherche. Il est possible de filtrer les produits “**informatique**” avec **un stock inférieur à 20** et les produits “**fourniture**” avec **un stock inférieur à 200** avec la recherche suivante :

```
SELECT * FROM produit
WHERE ( categorie = 'informatique' AND stock < 20 )
OR ( categorie = 'fourniture' AND stock < 200 )
```

Cela permet de retourner les 3 résultats suivants :

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	souris	informatique	16	30
4	crayon	fourniture	147	2

5 -3) Opérateur IN

Il s'utilise avec la commande **WHERE** pour vérifier si une colonne est égale à une des valeurs comprise dans un ensemble de valeurs déterminés. C'est une méthode simple pour vérifier si une colonne est égale à une valeur **OU** une autre valeur **OU** une autre valeur et ainsi de suite, sans avoir à utiliser de multiple fois l'opérateur **OR**.

Pour chercher toutes les lignes où la colonne "nom_colonne" est égale à 'valeur 1' OU 'valeur 2' ou 'valeur 3', on utilise la syntaxe suivante:

```
SELECT nom_colonne FROM nom_table  
WHERE nom_colonne IN ( valeur1, valeur2, valeur3, ... )
```

Cette syntaxe peut être associée à l'opérateur **NOT** pour rechercher toutes les lignes qui ne sont pas égales à l'une des valeurs stipulées.

```
WHERE nom_colonne NOT IN ( valeur1, valeur2, valeur3, ... )
```

LANGAGE D'INTERROGATION DES DONNÉES : LID

Exemple

Imaginons une table “adresse” qui contient une liste d’adresse associée à des utilisateurs d’une application.

id	id_utilisateur	addr_rue	addr_code_postal	addr_ville
1	23	35 Rue Madeleine Pelletier	25250	Bournois
2	43	21 Rue du Moulin Collet	75006	Paris
3	65	28 Avenue de Cornouaille	27220	Mousseaux-Neuville
4	67	41 Rue Marcel de la Provoté	76430	Graimbouville
5	68	18 Avenue de Navarre	75009	Paris

Si l’ont souhaite obtenir les enregistrements des adresses de Paris et de Graimbouville, il est possible d’utiliser la requête suivante:

```
SELECT * FROM adresse  
WHERE addr_ville IN ( 'Paris', 'Graimbouville' )
```


LANGAGE D'INTERROGATION DES DONNÉES : LID

Résultats :

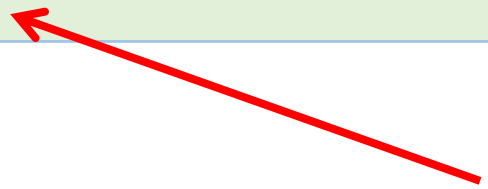
id	id_utilisateur	addr_rue	addr_code_postal	addr_ville
2	43	21 Rue du Moulin Collet	75006	Paris
4	67	41 Rue Marcel de la Provoté	76430	Graimbouville
5	68	18 Avenue de Navarre	75009	Paris

5 -4) Opérateur LIKE

Il s'utilise dans la clause **WHERE** et permet d'effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Les modèles de recherches sont multiple.

La syntaxe à utiliser pour utiliser l'opérateur LIKE est la suivante :

```
SELECT nom_colonne  
FROM nom_table  
WHERE nom_colonne LIKE 'A%'
```



Exemple de syntaxe de l'opérateur **LIKE**
permet de rechercher toutes les lignes de
"nom_colonne" qui **commence** par un **"A"**.

LANGAGE D'INTERROGATION DES DONNÉES : LID

- **LIKE '%a'** : le caractère “%” est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaînes de caractère qui se terminent par un “a”.
- **LIKE 'a%'** : ce modèle permet de rechercher toutes les lignes de “colonne” qui commencent par un “a”.
- **LIKE '%a%'** : ce modèle est utilisé pour rechercher tous les enregistrements qui utilisent le caractère “a”.
- **LIKE 'pa%on'** : ce modèle permet de rechercher les chaînes qui commencent par “pa” et qui se terminent par “on”, comme “pantalon” ou “pardon”.
- **LIKE 'a_c'** : peu utilisé, le caractère “_” (underscore) peut être remplacé par n'importe quel caractère, mais un seul caractère uniquement (alors que le symbole pourcentage “%” peut être remplacé par un nombre incalculable de caractères). Ainsi, ce modèle permet de retourner les lignes “aac”, “abc” ou même “azc”.

LANGAGE D'INTERROGATION DES DONNÉES : LID

Exemple :

Imaginons une table “client” qui contient les enregistrement d'utilisateurs :

id	nom	ville
1	Léon	Lyon
2	Odette	Nice
3	Vivien	Nantes
4	Etienne	Lille

Si l'ont souhaite obtenir uniquement les clients des villes qui commencent par un “N”, il est possible d'utiliser la requête suivante:

```
SELECT * FROM client  
WHERE ville LIKE 'N%'
```

LANGAGE D'INTERROGATION DES DONNÉES : LID

Résultat :

Avec cette requête, seul les enregistrements suivants seront retournés:

id	nom	ville
2	Odette	Nice
3	Vivien	Nantes

Obtenir les résultats terminent par “e” :

```
SELECT * FROM client  
WHERE ville LIKE '%e'
```

id	nom	ville
2	Odette	Nice
4	Etienne	Lille

6) La commande Order by

L'instruction **ORDER BY** dans SQL est utilisée pour trier les données extraites par ordre croissant ou décroissant selon une ou plusieurs colonnes.

- ❖ Par défaut, **ORDER BY** trie les données par ordre croissant.
- ❖ Vous pouvez utiliser le mot-clé **DESC** pour trier les données par ordre décroissant et le mot-clé **ASC** pour trier par ordre croissant.

Syntaxe :

```
SELECT nom_colonne FROM nom_table  
WHERE conditions  
ORDER BY nom_colonne ASC/DESC
```

7) Fonctions et opérateurs :

1) Opérateurs divers :

COUNT(*) / COUNT(Colonne)	Nombre d'enregistrements retournés par la sélection.
 	Concaténation de chaînes de caractères
+ - * /	Addition, soustraction, multiplication, division

2) Fonctions numériques :

ABS(n)	Valeur absolue ex: ABS(-27.6)=27.6
MOD(m,n)	Reste de la division de m par n , ex: MOD(35,4)=3
POWER(m,n)	m puissance n , ex: POWER(4,2)=16
SIGN(n)	Indique le signe de n , ex: SIGN(0)=0, SIGN(-5)=-1, SIGN(5)=1
SQRT(n)	Racine carrée de n , ex: SQRT(9)=3, SQRT(-9)=NULL
ROUND(n)	Arrondi de n , ex: ROUND(15.3)=15 ROUND(15.5)=16

7) Fonctions et opérateurs :

3) Fonctions de groupe/agrégation :

AVG(expr)	Moyenne de toutes les valeurs de expr
COUNT(*)	Nombre d'enregistrements retournés par la sélection.
COUNT(expr)	Nombre d'enregistrements retournés par la sélection, pour lesquels expr n'a pas une valeur NULL.
MAX(expr)	Valeur maximale de toutes les valeurs de expr
MIN(expr)	Valeur minimale de toutes les valeurs de expr
SUM(expr)	Somme de toutes les valeurs de expr

7) Fonctions et opérateurs :

4) Fonctions chaîne de caractères :

LENGTH(char)	Donne la longueur de la chaîne de caractères char. LENGTH('Intégrité')=9
INITCAP(char)	La première lettre de chaque mot de la chaîne de caractères est mise en majuscule.
LOWER(char)	Toutes les lettres sont mises en minuscules LOWER('ConFiAnce')='confiance'
UPPER(char)	Toutes les lettres sont mises en majuscules UPPER('ConFiAnce')='CONFIANCE'

7) Fonctions et opérateurs :

4) Fonctions de conversion :

Fonction	Description	Syntaxe Exemple
TO_NUMBER(exp)	Convertit une chaîne de caractères en nombre.	TO_NUMBER('123.45') donne 123.45 (nombre)
TO_CHAR(exp)	Convertit un nombre ou une date en chaîne de caractères.	TO_CHAR(123.45) donne '123.45' (texte)
TO_DATE(exp, format)	Convertit une chaîne de caractères en date selon un format donné.	TO_DATE('2023-01-01', 'YYYY-MM-DD') donne une date

7) Fonctions et opérateurs :

4) Fonctions de conversion :

Exemples d'utilisation :

Supposons que vous avez un champ **string_salary** dans la table employees qui contient le salaire sous forme de **chaîne de caractères** et que vous voulez le convertir en nombre pour le comparer.

```
SELECT employee_id, last_name, TO_NUMBER(string_salary) AS numeric_salary  
FROM employees  
WHERE TO_NUMBER(string_salary) > 5000;
```

8) Commande Group by

La commande GROUP BY en SQL permet d'organiser des données **identiques** en **groupes** à l'aide de certaines fonctions. C'est-à-dire si une colonne particulière a les mêmes valeurs dans différentes lignes, elle organisera ces lignes dans un groupe.

- ❖ La commande **GROUP BY** est utilisée avec l'instruction **SELECT**.
- ❖ Dans la requête, la commande **GROUP BY** est placée après la clause **WHERE**.
- ❖ Dans la requête, la commande **GROUP BY** est placée avant la clause **ORDER BY** si elle est utilisée.

Vous pouvez également utiliser certaines fonctions d'agrégation telles que **COUNT**, **SUM**, **MIN**, **MAX**, **AVG**, etc. sur la colonne groupée.

Syntaxe :

```
SELECT nom_colonne FROM nom_table  
WHERE conditions  
GROUP BY nom_colonne
```

8) Commande Group by

Exemple avec GROUP BY

Calculer le salaire moyen par département.

```
SELECT department_id, AVG(salary) AS average_salary  
FROM employees  
GROUP BY department_id;
```

6) Commande Having

La commande **HAVING** est utilisée après la commande **GROUP BY** pour poser des conditions afin de décider quel groupe fera partie de l'ensemble des résultats finaux.

De plus, nous ne pouvons pas utiliser les fonctions d'agrégation telles que **SUM()**, **COUNT()**, etc. avec la commande **WHERE**. Nous devons donc utiliser la commande **HAVING** si nous voulons utiliser l'une de ces fonctions dans les conditions.

Syntaxe :

```
SELECT nom_colonne FROM nom_table  
WHERE conditions  
GROUP BY nom_colonne  
HAVING conditions
```

8) Commande Having

Exemple avec GROUP BY avec Having

Trouver les départements où le salaire moyen est supérieur à 8000.

```
SELECT department_id, AVG(salary) AS average_salary  
FROM employees  
GROUP BY department_id;  
HAVING AVG(salary) > 8000;
```