# SI Cours 8 - SQL Partie 2

# 1- Introduction

- Le langage de manipulation de données SQL (DML) est utilisé pour interroger et modifier les données d'une base de données.
- Nous allons voir les commandes SQL DML:
  - SELECT: pour interroger les données dans la base de données.
  - INSERT: pour insérer des données dans une table.
  - UPDATE : pour mettre à jour des données dans une table.
  - DELETE: pour supprimer des données d'une table.

# 2- Instruction SELECT :

La commande SELECT permet d'extraire des données de tables selon des critères spécifiques. Elle est traitée selon la séquence suivante :

```
SELECT DISTINCT élément(s)
FROM table(s)
WHERE prédicat
GROUP BY champ(s)
ORDER BY champs;
```

- L'utilisation de DISTINCT est optionnelle:
  - elle est utilisée pour renvoyer uniquement des valeurs distinctes (différentes).
- On peut utiliser l'instruction SELECT pour retourner la liste de toutes les colonnes et de toutes les lignes de la table "employee":

```
select * from employee;
```

| idEmployee | First Name | Last Name | Phone_Number | Birthdate | Salary |
|---|---|---|---|---|---|
| 1000 | Alice | Smith | 123-456-7890 | 1990-01-01 | 50000 |
| 1001 | Bob | Johnson | 987-654-3210 | 1985-05-15 | 60000 |
| 1002 | Charlie | Brown | 555-555-5555 | 1978-08-20 | 55000 |
| 1003 | David | Lee | 111-222-3333 | 1992-12-10 | 48000 |
| 1004 | Emily | Davis | NULL | 1989-03-25 | 52000 |
| 1005 | Alicia | Smith | 222-333-4444 | 1995-07-12 | 45000 |
| 1006 | Robert | Johnson | 888-777-6666 | 1982-11-28 | 65000 |

| idEmployee | First Name | Last Name | Phone_Number | Birthdate | Salary |
|---|---|---|---|---|---|
| 1007 | Alice | Johnson | 333-444-5555 | 1988-02-15 | 58000 |
| 1008 | Robert | Johnson | 999-888-7777 | 1980-09-10 | 62000 |

On peut utiliser l'instruction SELECT pour retourner une liste téléphonique des employés à partir de la table Employees comme suit :

```
USE sw;
SELECT FirstName, LastName, Phone_Number
FROM Employee;
```

| First Name | Last Name | Phone_Number |
|---|---|---|
| Alice | Smith | 123-456-7890 |
| Bob | Johnson | 987-654-3210 |
| Charlie | Brown | 555-555-5555 |
| David | Lee | 111-222-3333 |
| Emily | Davis | NULL |
| Alicia | Smith | 222-333-4444 |
| Robert | Johnson | 888-777-6666 |
| Alice | Johnson | 333-444-5555 |
| Robert | Johnson | 999-888-7777 |

- On peut utiliser la commande SELECT pour retourner la liste de tous les prénoms (champ FirstName), mais en association au champ "FirstName" l'alias "The First Name"

```
SELECT FirstName as "The First Name"
FROM Employee;
```

| The First Name |
|---|
| Alice |
| Bob |
| Charlie |
| David |
| Emily |
| Alicia |
| Robert |

| The First Name |
| --- |
| Alice |
| Robert |

- L'instruction suivante permet de retourner la liste des prénoms uniques:

```
SELECT distinct FirstName as "The First Name"
FROM Employee;
```

| The First Name |
| --- |
| Alice |
| Alicia |
| Bob |
| Charlie |
| David |
| Emily |
| Robert |

- L'instruction suivante permet de retourner la liste des noms uniques:

```
SELECT distinct LastName
FROM Employee;
```

| Last Name |
| --- |
| Brown |
| Davis |
| Johnson |
| Lee |
| Smith |

- L'instruction suivante permet de retourner les noms et prénoms uniques

```
SELECT distinct Firstname, LastName
FROM Employee;
```

| First Name | Last Name |
|---|---|
| Alice | Johnson |
| Alice | Smith |
| Alicia | Smith |
| Bob | Johnson |
| Charlie | Brown |
| David | Lee |
| Emily | Davis |
| Robert | Johnson |

## 2-1 Critère WHERE dans la clause SELECT :

- La clause *where* sert à **limiter** la sélection d'enregistrements utilisant *BETWEEN, IN, NULL et NOT NULL*, ainsi que l'utilisation de caractères génériques dans la clause *LIKE*.

```
select FirstName from Employee
where idEmployee=1003
```

| First Name |
|---|
| David |

- *Exemple 1* utilise BETWEEN pour spécifier : idEmployee entre 1002 et 1005

```
SELECT FirstName, LastName
FROM Employee
WHERE idEmployee BETWEEN 1002 and 1005;
```

| First Name | Last Name |
|---|---|
| Charlie | Brown |
| David | Lee |
| Emily | Davis |
| Alicia | Smith |

- *Exemple 2*

```
SELECT FirstName, LastName
FROM Employee
```

```
WHERE idEmployee >=1002 and idEmployee<=1005;
```

| First Name | Last Name |
|---|---|
| Charlie | Brown |
| David | Lee |
| Emily | Davis |
| Alicia | Smith |

- *Exemple 3*

```
SELECT FirstName, LastName
FROM Employee
WHERE idEmployee >=1002 and salary<=50000;
```

| First Name | Last Name |
|---|---|
| David | Lee |
| Alicia | Smith |

- *Exemple 4* illustre comment limiter la sélection d'enregistrements avec le critère WHERE en utilisant NOT BETWEEN.

```
SELECT FirstName, LastName
FROM Employee
WHERE idEmployee NOT BETWEEN 1002 and 1007;
```

| First Name | Last Name |
|---|---|
| Alice | Smith |
| Bob | Johnson |
| Robert | Johnson |

- *Exemple 5*

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName = 'Alice' OR FirstName = 'Alicia' ;
```

| First Name | Last Name |
|---|---|
| Alice | Smith |

| First Name | Last Name |
|---|---|
| Alicia | Smith |
| Alice | Johnson |

- *Exemple 6* sélectionne des enregistrements en utilisant FirstName IN dans la clause WHERE.

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName IN ('Alice',  'Alicia');
```

| First Name | Last Name |
|---|---|
| Alice | Smith |
| Alicia | Smith |
| Alice | Johnson |

- Les deux derniers exemples illustrent comment NULL et NOT NULL peuvent être utilisés pour sélectionner des enregistrements
- *Exemple 7*: utilise NULL.

```
select FirstName, LastName from employee
where phone_number is null;
```

| First Name | Last Name |
|---|---|
| Emily | Davis |

- *Exemple 8*: utilise NOT NULL.

```
select FirstName, LastName from employee
where phone_number is not null;
```

| First Name | Last Name |
|---|---|
| Alice | Smith |
| Bob | Johnson |
| Charlie | Brown |
| David | Lee |
| Alicia | Smith |

| First Name | Last Name |
|---|---|
| Robert | Johnson |
| Alice | Johnson |
| Robert | Johnson |

## 2-2 Utilisation des caractères génériques dans la clause LIKE

- Le mot-clé *LIKE* sélectionne les lignes contenant des champs qui correspondent à des portions spécifiées de chaînes de caractères.
- *LIKE* est utilisé avec des données de type *char, varchar, text, datetime et smalldatetime*. Un caractère générique permet à l'utilisateur de faire correspondre des champs contenant certaines lettres.
- Le Tableau suivant montre quatre façons de spécifier des caractères génériques dans l'instruction SELECT au format d'expression régulière.

| Caractère | Description |
|---|---|
| % | N'importe quelle chaîne de zéro caractère ou plus |
| _ | N'importe quel caractère unique |
| [ ] | N'importe quel caractère unique dans la plage spécifiée (par exemple, [a-f]) ou l'ensemble (par exemple, [abcdef]) |
| [^] | N'importe quel caractère unique qui n'est pas dans la plage spécifiée (par exemple, [^a – f] ou l'ensemble (par exemple, [^abcdef]) |

- *exemple 1*: *LIKE 'Sm%'* recherche tous les noms de famille commençant par les lettres "Sm" .

```
SELECT FirstName, LastName FROM Employee
WHERE LastName LIKE 'Sm%';
```

| First Name | Last Name |
|---|---|
| Alice | Smith |
| Alicia | Smith |

- *exemple 2*: *LIKE '%on'* recherche tous les noms de famille se terminant par les lettres "on"

```
SELECT FirstName, LastName FROM Employee
WHERE LastName LIKE '%on';
```

| First Name | Last Name |
|---|---|
| Bob | Johnson |
| Robert | Johnson |
| Alice | Johnson |
| Robert | Johnson |

- *exemple 3*: *LIKE '%s%'* recherche tous les noms de famille contenant la lettre "s"

```
SELECT FirstName, LastName FROM Employee
WHERE LastName LIKE '%s%';
```

- Autre Exemples:

| First Name | Last Name |
|---|---|
| Alice | Smith |
| Bob | Johnson |
| Emily | Davis |
| Alicia | Smith |
| Robert | Johnson |
| Alice | Johnson |
| Robert | Johnson |

```
SELECT FirstName, LastName FROM Employee
WHERE LastName not LIKE '%s%';
```

| First Name | Last Name |
|---|---|
| Charlie | Brown |
| David | Lee |

```
SELECT FirstName, LastName FROM Employee
WHERE LastName LIKE '_%s%';
```

| First Name | Last Name |
|---|---|
| Bob | Johnson |
| Emily | Davis |
| Robert | Johnson |
| Alice | Johnson |
| Robert | Johnson |

```
SELECT FirstName, LastName FROM Employee
WHERE LastName not LIKE '_%s%';
```

| First Name | Last Name |
|---|---|
| Alice | Smith |
| Charlie | Brown |
| David | Lee |
| Alicia | Smith |

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName  LIKE '[cb]%';
```

| First Name | Last Name |
|---|---|
| Bob | Johnson |
| Charlie | Brown |

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName  LIKE '[b-d]%';
```

| First Name | Last Name |
|---|---|
| Bob | Johnson |
| Charlie | Brown |
| David | Lee |

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName  LIKE '[^a-f]%';
```

| First Name | Last Name |
|---|---|
| Robert | Johnson |
| Robert | Johnson |

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName  LIKE '[^abers]%';
```

| First Name | Last Name |
|---|---|
| Charlie | Brown |
| David | Lee |

## 2-3 L'instruction SELECT avec clause ORDER BY :

Utilisez la clause *ORDER BY* pour trier les enregistrements dans la liste résultante. Utilisez ASC pour trier les résultats par ordre croissant et DESC pour trier les résultats par ordre décroissant.
Par exemple, avec ASC :

```
select idemployee from employee
where idemployee>1005
order by birhdate asc;
```

| idEmployee |
|---|
| 1008 |
| 1006 |
| 1007 |

Et avec DESC :

```
select idemployee from employee
where idemployee>1005
order by birhdate desc;
```

| idEmployee |
|---|
| 1007 |
| 1006 |

| idEmployee |
| --- |
| 1008 |

```
select idemployee from employee
where firstname like '%b%'
order by firstname asc, birthdate asc ;
```

| idEmployee |
| --- |
| 1001 |
| 1008 |
| 1006 |

## 2-4 Instruction SELECT avec clause GROUP BY :

La clause *GROUP BY* est utilisée pour créer une ligne de sortie par groupe et produit des valeurs de résumé pour les colonnes sélectionnées.

```
SELECT  lastname
FROM employee WHERE idemployee > 1004
GROUP BY lastname;
```

| lastname |
| --- |
| Johnson |
| Smith |

### 2-4-1 Utilisation de COUNT avec GROUP BY :

*COUNT* peut être utilisée pour dénombrer le nombre d'éléments dans un conteneur. Si vous souhaitez compter différents éléments dans des groupes distincts, tels que des billes de couleurs variées, vous utiliseriez la fonction COUNT avec la commande GROUP BY.

```
SELECT  COUNT(*) as "Number of Employees" FROM employee;
```

| Number of Employees |
| --- |
| 9 |

```
SELECT  COUNT(Lastname) as "Number of Employees" FROM employee;
```

| Number of Employees |
| --- |
| 9 |

```
SELECT  COUNT(distinct LastName) as "Number of Unique Lastnames " FROM
employee;
```

| Number of Unique Lastnames |
| --- |
| 5 |

```
SELECT   LastName,  COUNT(Lastname) as "Occurrences"
FROM employee
where idEmployee>1004
group by LastName;
```

| LastName | Occurrences |
| --- | --- |
| Johnson | 3 |
| Smith | 1 |

## 2-4-2 Utilisation de AVG et SUM avec GROUP BY :

*AVG* donne la moyenne d'un groupe, et *SUM* donne le total. Voici des exemples utilisant ces fonctions avec la clause GROUP BY.

```
SELECT   LastName,  sum(salary) as "Sum Salary"
FROM employee
where idEmployee>1004
group by LastName;
```

| LastName | Sum Salary |
| --- | --- |
| Johnson | 185000 |
| Smith | 45000 |

```
SELECT   LastName,  avg(salary) as "Avg Salary"
FROM employee
```

```
where idEmployee>1004
group by LastName;
```

| LastName | Avg Salary |
|---|---|
| Johnson | 61666.6666666667 |
| Smith | 45000 |

```
SELECT   LastName,  sum(salary)/3 as "Sum Salary by 3"
FROM employee
where idEmployee>1004
group by LastName;
```

| LastName | Sum Salary by 3 |
|---|---|
| Johnson | 61666.6666666667 |
| Smith | 15000 |

## 2-4-3 Restriction des lignes avec HAVING :

La clause *HAVING* peut être utilisée pour restreindre les lignes, similaire à la condition WHERE, mais applicable aux groupes.

```
SELECT   LastName,  sum(salary) as 'Sum Salary'
from Employee
where idEmployee>1004
group by LastName
having sum(salary)> 50000;
```

| LastName | Sum Salary |
|---|---|
| Johnson | 185000 |

```
SELECT   LastName,  sum(salary) as 'Sum Salary'
from Employee
where idEmployee>1004
group by LastName
having LastName like 'S%';
```

| LastName | Sum Salary |
|---|---|
| Smith | 45000 |

# 3- Fonctions intégrées

Il existe de nombreuses fonctions intégrées dans SQL Server, telles que :

1. Agrégation : renvoie des valeurs récapitulatives
2. Conversion : transforme un type de données en un autre
3. Date : affiche des informations sur les dates et les heures
4. Mathématiques : effectue des opérations sur des données numériques
5. Chaîne : effectue des opérations sur des chaînes de caractères, des données binaires ou des expressions
6. Système : renvoie une information spéciale de la base de données
7. Texte et image : effectue des opérations sur des données texte et image

## 3-1 Fonctions d'agrégation

Les fonctions d'agrégation effectuent un calcul sur un ensemble de valeurs et renvoient une seule valeur récapitulative.

| FONCTION | DESCRIPTION |
|---|---|
| AVG | Renvoie la moyenne de toutes les valeurs, ou uniquement des valeurs DISTINCT, dans l'expression. |
| COUNT | Renvoie le nombre de valeurs non nulles dans l'expression. Lorsque DISTINCT est spécifié, COUNT trouve le nombre de valeurs non nulles uniques. |
| COUNT(*) | Renvoie le nombre de lignes. COUNT(*) ne prend aucun paramètre et ne peut pas être utilisé avec DISTINCT. |
| MAX | Renvoie la valeur maximale dans l'expression. MAX peut être utilisé avec des colonnes numériques, de caractères et de datetime, mais pas avec des colonnes bit. Avec les colonnes de caractères, MAX trouve la valeur la plus élevée dans la séquence de tri. MAX ignore les valeurs nulles. |
| MIN | Renvoie la valeur minimale dans l'expression. MIN peut être utilisé avec des colonnes numériques, de caractères et de datetime, mais pas avec des colonnes bit. Avec les colonnes de caractères, MIN trouve la valeur la plus basse dans la séquence de tri. MIN ignore les valeurs nulles. |
| SUM | Renvoie la somme de toutes les valeurs, ou uniquement des valeurs DISTINCT, dans l'expression. SUM ne peut être utilisé qu'avec des colonnes numériques. |

Ci-dessous se trouvent des exemples de chacune des fonctions d'agrégation répertoriées dans le tableau

**Exemple 1 : AVG**

```
SELECT AVG(Salary) AS 'Average Salary'
FROM employee;
```

| Average Salary |
| --- |
| 55000 |

**Exemple 2 : COUNT**

```
SELECT count(phone_number) as "Number Phone Numbers"
FROM employee;
```

| Number Phone Numbers |
| --- |
| 8 |

**Exemple 3 :

```
    SELECT count(lastname) as "Number Last names"
    FROM employee;
```

| Number Last names |
| --- |
| 9 |

```
    SELECT count(distinct lastname) as "Number Last names"
    FROM employee;
```

| Number Last names |
| --- |
| 5 |

**Exemple 4 : MAX**

```
    SELECT max(salary) as 'Max Salary'
    FROM employee;
```

| Max Salary |
| --- |
| 65000 |

```
SELECT max(birthDate) as 'Max Birth Date'
FROM employee;
```

| Max Birth Date |
| --- |
| 1995-07-12 |

```
SELECT max(birthDate) as 'Max Birth Date'
FROM employee
where idemployee >1008;
```

| Max Birth Date |
| --- |
| NULL |

```
SELECT max(lastname) as 'Max Last Name'
FROM employee;
```

| Max Last Name |
| --- |
| Smith |

**Exemple 5 : MIN**

```
SELECT min(salary) as 'Min Salary'
FROM employee;
```

| Min Salary |
| --- |
| 45000 |

```
SELECT min(birthDate) as 'Min Birth Date'
FROM employee;
```

| Min Birth Date |
| --- |
| 1978-08-20 |

```
    SELECT min(birthDate) as 'Min Birth Date'
    FROM employee
    where idemployee >1008;
```

| Min Birth Date |
| --- |
| NULL |

```
select  min(phone_number) as "min phone num"
from employee;
```

| min phone num |
| --- |
| 111-222-3333 |

```
    SELECT min(lastname) as 'Min Last Name'
    FROM employee;
```

| Min Last Name |
| --- |
| Brown |

**Exemple 6 : SUM**

```
SELECT SUM(salary) AS 'Sum Salary' FROM Employee;
```

| Sum Salary |
| --- |
| 495000 |

```
SELECT idemployee
FROM Employee
WHERE phone_number = (SELECT min(phone_number) FROM employee)
```

| idemployee |
| --- |
| 1003 |

## 3-2 Fonction de conversion

Les fonctions de conversion transforment un type de données en un autre.

- Syntaxe de la fonction Cast:

```
CAST ( expression AS data_type [ ( length ) ] )
```

- Syntaxe de la fonction Convert:

```
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

- Exemples:

```
select idemployee
from employee
where convert(char(4),idEmployee) like '%3%';
```

```
select idemployee
from employee
where cast(idEmployee as char(4)) like '%3%';
```

| idEmployee |
| --- |
| 1003 |

- La requête suivante donne le même résultat:

```
select idEmployee
from employee
where idEmployee like '%3%';
```

| idEmployee |
| --- |
| 1003 |

## 3-3 Fonctions de date

- Exemples de fonctions de manipulation de dates:
  - DATEADD (datepart, number, date ): Retourne une nouvelle valeur datetime en ajoutant un intervalle de durée (number * datepart spécifié) à la date spécifiée
  - DATEDIFF ( datepart, startdate, enddate ): Renvoie le nombre de limites de date ou d'heure de datepart, franchies entre deux dates spécifiées.

    - GETEDATE () : Renvoie une valeur datetime contenant la date et l'heure de l'ordinateur sur lequel l'instance de SQL Server s'exécute. La valeur renvoyée n'inclut pas le décalage horaire."
    - ISDATE(): Détermine si une expression d'entrée de type datetime ou smalldatetime a une valeur de date ou d'heure valide.
    - DAY(date): Renvoie un entier représentant la partie jour de la date spécifiée
    - MONTH(date): Renvoie un entier représentant la partie mois d'une date spécifiée.
    - YEAR (date): Renvoie un entier représentant la partie année d'une date spécifiée
- **Exemples d'utilisation**

```
SELECT DATEADD(MONTH, 3, birthDate) as "Add 3 months"
FROM Employee
where idEmployee=1000;
```

| Add 3 months |
| --- |
| 1990-04-01 |

- Valeurs possible pour le paramètre datepart:

| DATEPART | Remarque |
| --- | --- |
| Year | |
| Quarter | |
| Month | |
| DAY | |
| WEEK | |
| HOUR | type du troisième paramètre DATETIME ou TIME avec dateadd |
| MINUTE | type du troisième paramètre DATETIME ou TIME avec dateadd |
| SECOND | type du troisième paramètre DATETIME ou TIME avec dateadd |
| MILLISECOND | type du troisième paramètre DATETIME ou TIME avec dateadd |

```
SELECT DATEDIFF(YEAR, birthdate, '2024-11-17' ) as 'Age'
FROM Employee
where idEmployee=1000;
```

| Age |
| --- |
| 35 |

```
SELECT  GETDATE() as 'Today';
```

| Today |
| --- |
| 2025-11-25 09:35:19.337 |

```
SELECT DATEDIFF(YEAR, birthdate,getdate()) as 'Age'
FROM Employee
where idEmployee=1000;
```

| Age |
| --- |
| 35 |

```
SELECT day(birthdate) as "day"
FROM Employee
where idEmployee=1001;
```

| Day |
| --- |
| 15 |

```
SELECT month(birthdate) as "Month"
FROM Employee
where idEmployee=1001;
```

| Month |
| --- |
| 5 |

```
SELECT year(birthdate) as "Year"
FROM Employee
where idEmployee=1001;
```

| Year |
| --- |
| 1985 |

```sql
select isdate('20:30') as 'validation';
```

```sql
select isdate('2024-01-01') as 'validation';
```

```sql
select isdate('2024-01-01 20:30') as 'validation';
```

| validation |
| --- |
| 1 |

```sql
select idEmployee, isdate(convert(datetime, birthdate)) as 'valdiation'
from Employee
where idEmployee=1000;
```

## 3-4 Fonctions mathématiques

- Les fonctions mathématiques effectuent des opérations sur des données numériques.

```sql
SELECT idemployee, salary, (salary * 1.1) AS 'New Salary'
from Employee
where idemployee = '1000';
```

| idemployee | salary | New Salary |
| --- | --- | --- |
| 1000 | 50000 | 55000 |

```sql
SELECT SQRT(81) as 'Square root';
```

| Square root |
| --- |
| 9 |

```sql
SELECT   LastName,  round(avg(salary),2) as "Avg Salary"
FROM employee
where idEmployee>1004
group by LastName;
```

| LastName | Avg Salary |
|---|---|
| Johnson | 61666.67 |
| Smith | 45000 |

```
use sw;
select idEmployee, round(sqrt(salary),2) as "square root"
from employee
where idEmployee>1006;
```

| idEmployee | Square root |
|---|---|
| 1007 | 240.83 |
| 1008 | 249 |

```
SELECT  ROUND(4567.2376,2) as "rounded number";
```

| Rounded Number |
|---|
| 4567.2400 |

```
SELECT  floor(4567.2376) as "floor int";
```

| floor int |
|---|
| 4567 |

```
SELECT idEmployee, floor(4567.2376) as "floor int"
from employee where idemployee >1006;
```

| idEmployee | floor int |
|---|---|
| 1007 | 4567 |
| 1008 | 4567 |

```
select ceiling(4567.2376) as 'ceiling int';
```

| ceiling int |
| --- |
| 4568 |

- Autre fonctions mathématiques:
  - ABS(): retourne la valeur absolue
  - PI(): retourne la valeur de PI
  - LOG():Renvoie le logarithme naturel de l'expression flottante spécifiée dans SQL Server.
  - COS(): renvoie le cosinus trigonométrique de l'angle spécifié - mesuré en radians - dans
    l'expression spécifiée
    EXP(): Renvoie la valeur exponentielle de l' expression flottante spécifiée .
- **Exemples**

```
select abs(-305) as 'abs';
```

| abs |
| --- |
| 305 |

```
select PI() as 'PI';
```

| PI |
| --- |
| 3.14159265358979 |

```
select Log(205) as 'Log';
```

| abs |
| --- |
| 305 |

```
select cos(pi()/3) as 'cos';
```

| cos |
| --- |
| 0.5 |

```
SELECT EXP(LOG(20)) as 'val';
```

| val |
| --- |
| 20 |

## 3-5 Fonction de chaîne

- TRIM(): Supprime le caractère d'espace char(32) ou d'autres caractères spécifiés du début et de la fin d'une chaîne.
    - À partir de SQL Server 2022 (16.x), supprime de manière optionnelle le caractère d'espace char(32) ou d'autres caractères spécifiés du début, de la fin ou des deux côtés d'une chaîne.
- UPPER(): Renvoie une expression de caractère avec des données de caractère en minuscules converties en majuscules
- LOWER(): Renvoie une expression de caractère après avoir converti les données de caractère en majuscules en minuscules.
- CONCAT(): renvoie une chaîne résultant de la concaténation, ou de l'assemblage, de deux ou plusieurs valeurs de chaîne de manière continue
- SUBSTRING(): Renvoie une partie d'une expression de caractère, binaire, de texte ou d'image dans SQL Server.
- REPLACE(): Remplace toutes les occurrences d'une valeur de chaîne spécifiée par une autre valeur de chaîne.
- LEN(): Renvoie le nombre de caractères de l'expression de chaîne spécifiée, à l'exclusion des espaces de fin.

```
SELECT  trim('     this is trimming     ') as 'trim';
```

| trim |
| --- |
| this is trimming |

```
SELECT  trim('t' from 'this is trimming') as 'trim';
```

| trim |
| --- |
| his is trimming |

```
SELECT upper(firstname) as 'upper'
from employee
```

```
where idEmployee=1008;
```

| upper |
| --- |
| ROBERT |

```
SELECT lower(firstname) as 'lower'
from employee
where idEmployee=1008;
```

| lower |
| --- |
| robert |

```
SELECT  concat(trim(firstname),' ',trim(LastName)) as 'name'
from employee
where idEmployee=1008;
```

| concat |
| --- |
| Robert Johnson |

```
SELECT  SUBSTRING(firstname,1,5) as 'substring'
from employee
where idEmployee=1006;
```

| substring |
| --- |
| Rober |

```
SELECT  replace(FirstName,'R','J')  as 'replace'
from employee
where idEmployee=1006;
```

| replace |
| --- |
| Jobejt |

```
SELECT
concat(replace(substring(FirstName,1,1),'R','J'),substring(firstname,
```

```
2,len(firstname)-1))  as 'replace'
from employee
where idEmployee=1006;
```

| replace |
| --- |
| Jobert |

```
select len('expression') as len;
```

| len |
| --- |
| 10 |

```
select len(firstname) as len
from Employee
where idEmployee>1005
```

| len |
| --- |
| 6 |
| 5 |
| 6 |

# 4- Jointure de tables

Joindre deux tables ou plus consiste à comparer les données dans des colonnes spécifiées et à utiliser
les résultats de la comparaison pour former une nouvelle table à partir des lignes qui qualifient. Une opération *JOIN* :

- Spécifie une colonne de chaque table
- Compare les valeurs dans ces colonnes ligne par ligne
- Combine les lignes avec des valeurs qualifiantes en une nouvelle ligne
  Bien que la comparaison soit généralement pour l'égalité - des valeurs qui correspondent exactement - d'autres types de joints peuvent également être spécifiés.
- Pour les sections qui suivent nous allons travailler avec ces 3 tables:
- **La table Livre:**

| ID_Livre | Titre | ID_Auteur | anné_pub | Langue | ID_Catégorie |
| --- | --- | --- | --- | --- | --- |
| 1 | Le Petit Prince | 7 | 1943 | Français | 3 |

| ID_Livre | Titre | ID_Auteur | anné_pub | Langue | ID_Catégorie |
|---|---|---|---|---|---|
| 2 | 1984 | 2 | 1949 | Anglais | 1 |
| 3 | Le Seigneur des Anneaux | 9 | 1954 | Anglais | 2 |
| 4 | Moby Dick | 4 | 1851 | Anglais | 4 |
| 5 | Les Misérables | 3 | 1862 | Français | 5 |
| 6 | Pride and Prejudice | 6 | 1813 | Anglais | 4 |
| 7 | Le Nom de la Rose | 8 | 1980 | Italien | 6 |
| 8 | Harry Potter à l'école des sorciers | 1 | 1997 | Anglais | 2 |
| 9 | Dune | 5 | 1965 | Anglais | 1 |

- **La table Catégorie**

| ID | Catégorie | Sous-catégorie |
|---|---|---|
| 1 | Littérature | Science-fiction |
| 2 | Littérature | Fantasy |
| 3 | Littérature | Littérature jeunesse |
| 4 | Littérature | Roman |
| 5 | Littérature | Roman historique |
| 6 | Littérature | Mystère |
| 7 | Science | Physique |
| 8 | Science | Chimie |
| 9 | Histoire | Antiquité |
| 10 | Histoire | Moyen Âge |

- **La Table Auteur**

| id_auteur | nom | prenom |
|---|---|---|
| 1 | Rowling | J.K. |
| 2 | Orwell | George |
| 3 | Hugo | Victor |
| 4 | Melville | Herman |
| 5 | Herbert | Frank |
| 6 | Austen | Jane |

| id_auteur | nom | prenom |
|---|---|---|
| 7 | Saint-Exupéry | Antoine |
| 8 | Eco | Umberto |
| 9 | Tolkien | J.R.R. |

- Les tables ont été crées par les requêtes suivantes:
- catégorie:

```
create table catégorie(
id_catégorie int not null primary key,
catégorie char(50) not null,
sous_catégorie char(50) not null,

)
```

- auteur:

```
create table auteur(
id_auteur int not null primary key,
nom char(50) not null,
prénom char(50) not null,
)
```

- livre

```
create table livre(
id_livre int not null primary key,
titre char(100) not null,
id_auteur int not null,
anné_pub int not null,
langue char(50),
id_catégorie int not null,
constraint fk_auteur foreign key(id_auteur) references Auteur(id_auteur),
constraint fk_catégorie foreign key (id_catégorie) references Catégorie (id)

)
```

## 4-1 Inner Join

Une jointure interne connecte deux tables sur une colonne du même type de données. Seules les lignes où les valeurs de la colonne correspondent sont renvoyées ; les lignes non appariées sont ignorées.

- **Exemple**

- Par exemple si on veut publier pour chaque sous catégorie qui a un identifiant supérieur ou égal à 5, le titre du livre correspondant. Ne prendre en considération que les sous-catégorie qui ont un titre. Les titres qui n'ont pas de catégorie correspondante ne sont pas affichés aussi
- Voici comment le faire avec Inner Join:

```
SELECT catégorie.sous_catégorie,  livre.titre
FROM catégorie
INNER  JOIN   livre
ON catégorie.id_catégorie = livre.id_catégorie
where catégorie.id_catégorie >=5
```

- Le résultat de la requête serait:

| sous-catégorie | titre |
|---|---|
| Roman Historique | Les Misérables |
| Mystère | Le Nom de La Rose |

- Le même résultat pourrait être obtenu en utilisant sans JOIN comme suit:

```
SELECT catégorie.sous_catégorie,  livre.titre
FROM catégorie, livre
where catégorie.id_catégorie = livre.id_catégorie
and catégorie.id_catégorie >=5
```

## 4-2 Left outer join

Une jointure externe gauche spécifie que toutes les lignes externes gauches doivent être renvoyées. Toutes les lignes de la table de gauche qui n'ont pas satisfait à la condition spécifiée sont incluses dans l'ensemble de résultats, et les colonnes de sortie de l'autre table sont définies sur NULL.

- **Exemple**
- On cherche à afficher pour chaque sous catégorie qui a un identifiant >=5, les titres des livres publiés. Si la sous catégorie n'a pas de titre correspondant, la sous catégorie sera quand même publiée.
- Pour ceci, on peut utiliser la requête suivante:

```
SELECT catégorie.sous_catégorie,  livre.titre
FROM catégorie
LEFT OUTER JOIN   livre
```

```
ON catégorie.id_catégorie = livre.id_catégorie
where catégorie.id_catégorie >=5
```

- Et le résultat serait:

| sous-catégorie | titre |
|---|---|
| Roman Historique | Les Misérables |
| Mystère | Le Nom de La Rose |
| Physique | NULL |
| Chimie | NULL |
| Antiquité | NULL |
| Moyen Âge | NULL |

- On peu aussi afficher le nombre des titres par sous-catégorie pour les catégorie qui ont un identifiant >=5. La requête serai:

```
SELECT catégorie.sous_catégorie, count(Livre.titre) as 'Nombre'
FROM catégorie
LEFT OUTER JOIN   livre
ON catégorie.id_catégorie = livre.id_catégorie
where catégorie.id_catégorie >=5
group by catégorie.sous_catégorie;
```

- Et Le résultat serait

| sous-catégorie | Nombre |
|---|---|
| Antiquité | 0 |
| Chimie | 0 |
| Moyen Âge | 0 |
| Mystère | 1 |
| Physique | 0 |
| Roman Historique | 1 |

## 4-3 Right outer join

Une jointure externe droite inclut, dans son ensemble de résultats, toutes les lignes de la table de droite qui n'ont pas satisfait à la condition spécifiée. Les colonnes de sortie qui correspondent à l'autre table sont définies sur NULL.

**Exemple**

Par exemple si on veut publier pour chaque sous catégorie qui a un identifiant supérieur ou égal à 5, le titre du livre correspondant. Ne prendre en considération que les sous-catégorie qui ont un titre. Mais si il y a un titre qui n'a pas de sous catégorie correspondante, le titre sera affiché

```
SELECT catégorie.sous_catégorie,  livre.titre
FROM catégorie
RIGHT OUTER JOIN   livre
ON catégorie.id_catégorie = livre.id_catégorie
where catégorie.id_catégorie >=5
```

- Dans notre cas, il n'y a pas de titre qui n'a pas de sous catégorie, donc on aura le même résultat que pour le INNER JOIN:

| sous-catégorie | titre |
|---|---|
| Roman Historique | Les Misérables |
| Mystère | Le Nom de La Rose |
| Physique | NULL |
| Chimie | NULL |
| Antiquité | NULL |
| Moyen Âge | NULL |

## 4-4 Full outer join

Une jointure externe complète spécifie que si une ligne de l'une ou l'autre table ne correspond pas aux critères de sélection, la ligne est incluse dans l'ensemble de résultats, et ses colonnes de sortie qui correspondent à l'autre table sont définies sur NULL.

**Exemple**

```
SELECT catégorie.sous_catégorie,  livre.titre
FROM catégorie
FULL OUTER JOIN   livre
ON catégorie.id_catégorie = livre.id_catégorie
where catégorie.id_catégorie >=5
```

- Dans notre cas, il n'y a pas de titre qui n'a pas de sous catégorie, donc on aura le même résultat que pour le LEFT OUTER JOIN:

| sous-catégorie | titre |
|---|---|
| Antiquité | 0 |
| Chimie | 0 |
| Moyen Âge | 0 |
| Mystère | 1 |
| Physique | 0 |
| Roman Historique | 1 |

## 4-5 Cross Join

Une jointure croisée est un produit combinant deux tables. Cette jointure renvoie les mêmes lignes que si aucune clause WHERE n'était spécifiée. Par exemple :

```
SELECT  livre.titre, auteur.nom  FROM livre
CROSS JOIN auteur
where id_livre>=7
and auteur.id_auteur>=8;
```

| titre | nom |
|---|---|
| Le Nom de La Rose | Eco |
| Harry Potter à l'école des sorciers | Eco |
| Dune | Eco |
| Le Nom de La Rose | Tolkien |
| Harry Potter à l'école des sorciers | Tolkien |
| Dune | Tolkien |

# 5- Instruction INSERT :

l'instruction INSERT ajoute des lignes à une table. Voici un exemple d'utilisation de l'instruction INSERT pour ajouter deux enregistrements à la table Auteur.

```
INSERT INTO auteur(id_auteur, nom, prénom)
VALUES(10, 'Flaubert', 'Gustave'),
       (11, 'Camus', 'Albert');
```

Pour insérer des lignes dans une table avec une colonne *IDENTITY,* les valeurs pour la colonne Identity ne doivent pas être incluses.

- On peut utiliser l'instruction Insert en combinaison avec l'instruction select pour copier des valeurs d'une table vers une autre:
- **Exemples**

```
INSERT INTO test2 (id_test2, val2, date2)
SELECT id_test1, val1, date1 FROM test1;
```

# 6- Instruction UPDATE

l'instruction *UPDATE* modifie les données dans les lignes existantes dans une table (ou une vue).
Cet exemple utilise l'instruction *UPDATE* pour mettre à jour le champ phone_number pour la ligne pour laquelle la valeur de phone_number est NULL:

```
USE sw;
UPDATE Employee
SET phone_number = '555-555-1212'
where phone_number is NULL
```

- Si on veut donner la même valeur pour toute les ligne, il suffit d'enlever la clause WHERE

Cet exemple augmente le montant des salaires de 10 % pour les salaires entre 40000 et 50000

```
use sw;
UPDATE Employee
SET salary = salary + (salary * 0.1)
WHERE salary BETWEEN 40000 and 50000;
```

# 6-1 Inclusion de sous-requêtes dans une requête UPDATE

- dans l'exemple suivant, les livres qui ont un titre qui contient "Harry Potter" dans la table livres, auront la valeur sous_catégorie dans la table Catégorie changée à "Fantasy-Fiction".

```
Use Library
Update Catégorie
SET sous_catégorie = 'Fantasy-Fiction'
WHERE id_catégorie IN (SELECT id_catégorie FROM Livre WHERE titre like
'%Harry Potter%');
```

# 7- Instruction DELETE

L'instruction *DELETE* supprime des lignes d'un ensemble d'enregistrements. *DELETE* nomme la table ou la vue qui contient les lignes qui seront supprimées, et une seule table ou ligne peut être répertoriée à la fois. *WHERE* est une clause *WHERE* standard qui limite la suppression à des enregistrements sélectionnés.

La syntaxe DELETE ressemble à ceci.

```
DELETE [FROM] {table_name | view_name} [WHERE clause]
```

Les règles pour l'instruction DELETE sont :

1. Si vous omettez une clause WHERE, toutes les lignes de la table sont supprimées (à l'exception des index, de la table, des contraintes).
2. DELETE ne peut pas être utilisé avec une vue qui a une clause FROM nommant plus d'une table. (Delete ne peut affecter qu'une seule table de base à la fois.)

Voici trois requêtes DELETE différentes qui peuvent être utilisées.

1. Suppression de toutes les lignes d'une table.

```
Use SW;
DELETE FROM Employee
```

2. Suppression de lignes sélectionnées :

```
use Library;
DELETE FROM Auteur
WHERE id_auteur=11;
```

3. Suppression de lignes basées sur une valeur dans une sous-requête :

- La requête supprime les auteurs qui n'on pas de livre dans la table livre

```
Use library;
DELETE FROM Auteur
WHERE id_auteur NOT IN (SELECT distinct id_auteur FROM livre);
```

# 8- Exemples Avancés de manipulation d'information avec l'instruction Select:

Dans la plupart des exemple, nous utiliserons la table Employee de la base de données sw décrite précédemment:

## 8-1 Combiner plusieurs instructions Select

- En prenant en considération la table Employee de la base de données sw, décrites précédemment, on veut connaitre les noms et prénoms des employés qui ont le plus grand salaire. Alors on combine deux instruction SELECT:
  - La requête incluse nous permet de connaître la valeur du plus grand salaire.
  - L'autre requête nous permet de connaître les id, nom, prénoms et salaire des employées qui ont un salaire égal au salaire maximum:
- La requête SQL serait:

```sql
select idEmployee, firstname, lastname, Salary
from Employee
where salary = (select max(salary) from employee);
```

- Le résultat serait:

| idEmployee | firstname | lastname | Salary |
|---|---|---|---|
| 1006 | Robert | Johnson | 65000 |

## 8-2 Utilisation de l'expression CASE

- L'expression 'CASE' évalue une liste de conditions et renvoie l'une des multiples expressions de résultat possibles.
- Elle peut être sous deux formes:
  - L'expression simple `CASE` compare une expression à un ensemble d'expressions simples pour déterminer le résultat.
  - L'expression recherchée `CASE` évalue un ensemble d'expressions booléennes pour déterminer le résultat.
- **Syntaxe**

```sql
-- expression CASE Simple:
CASE input_expression
     WHEN when_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END

-- expression CASE Recherchée:
CASE
     WHEN Boolean_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```

**Exemples**

- En prenant en considération la table Employee de la base de données sw, décrites précédemment, on veut créer un annuaire téléphonique contentent les id, noms, prénoms, et numéros de téléphones des employés. Pour les employés qui n'ont pas de valeur de numéro de téléphone définie, on retourne la phrase "No Phone Number".
- La requête serait:

```
select idemployee, firstname, lastname,
case
when phone_number is not null then phone_number
else 'No Phone Number'
end as 'Phone Number'
from Employee;
```

- ou bien:

```
select idemployee, firstname, lastname,
case len(phone_number)
when 12 then phone_number
else 'No Phone Number'
end as 'Phone Number'
from Employee;
```

## 8-3 Extraire des informations sur les bases de données

- Pour connaître la liste de toutes les bases de données d'un SQL server;

```
select * from sys.databases;
```

- Pour connaître seulement les noms des bases de données existantes:

```
select name from sys.databases;
```

- pour retourner les noms, id, et dates de création des bases de données existantes:

```
SELECT name, database_id, create_date
FROM sys.databases;
```

- pour retourner la liste des nom des tables dans une base données (dans l'exemple: BD= sw):

```
use sw;
SELECT TABLE_NAME
```

```
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
```

- pour retourner la liste des noms des vues dans une base de données (dans l'exemple: BD= sw):

```
use sw;
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'VIEW'
```

- pour retourner la liste des noms de colonnes , ainsi que leur type de la table "Employee":

```
USE sw;
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Employee';
```

- pour retourner la liste des contraintes d'une table (la table Employee de la BD: sw):

```
USE sw;
SELECT CONSTRAINT_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'Employee';
```

## 8-4 La création d'une vue

- Une vue est une table virtuelle dont le contenu est défini par une requête.
- Comme une table, une vue se compose d'un ensemble de colonnes et de lignes de données nommées.
- Les lignes et les colonnes de données proviennent de tables référencées dans la requête définissant la vue et sont produites de manière dynamique lorsque la vue est référencée.
- Pour créer une vue contenant seulement le id des employées, leur prénom et nom, et leur numéro de téléphone, l'une des deux requêtes suivantes peut être utilisé:

```
use sw
go
CREATE VIEW Phone_Directory
AS
select idEmployee, firstname, LastName, Phone_Number
from Employee;
```

- ou bien:

```
CREATE VIEW Phone_Directory
AS
select idEmployee, firstname, LastName, Phone_Number
from sw.dbo.Employee;
```

- Il est possible de sélectionner les valeurs de la table comme suit:

```
use sw;
select * from phone_directory;
```

# 8-5 Combiner Plusieurs Jointure

Il est possible de combiner plusieurs jointures, pour pouvoir extraire l'information requise. Par exemple, en prenant en considération les tables suivantes (appartenant à la base de données Ecommerce)

**Produits**(IDProduit, Nom, Description, Prix, QuantiteEnStock, IDCategorie)
**Clients**(IDClient, Nom, Prenom, Adresse, Email, Telephone )
**Commandes**(IDCommande, IDClient, DateCommande,EtatCommande)
**Catégorie**(IDCategorie, Nom)
**LigneCommande**(IDLigneCommande, IDCommande, IDProduit, Quantité)

Sachant que dans:

- **Table Produits:**
  - IDCategorie : clé étrangère vers la table Categories)
- **Commandes:**
  - IDClient :clé étrangère vers la table Clients
  - EtatCommande: peut prendre l'une des valeurs suivantes: en cours, expédiee, livree
- **LigneCommande:**
  - IDCommande : clé étrangère vers la table Commandes
  - IDProduit : clé étrangère vers la table Produits
- On peut combiner deux instructions de Inner Join, pour pouvoir connaître les identifiants et les dates des commandes dont la valeur totale dépasse 5000 €

```
  use Ecommerce;

SELECT LignesCommande.IDCommande, Commandes.DateCommande ,
SUM(LignesCommande.Quantite * Produits.Prix) AS TotalCommandes
FROM LignesCommande
INNER JOIN Commandes  ON Commandes.IDCommande =
LignesCommande.IDCommande
INNER JOIN Produits  ON LignesCommande.IDProduit = Produits.IDProduit
```

```
GROUP BY LignesCommande.IDCommande, commandes.DateCommande
HAVING SUM(LignesCommande.Quantite * Produits.Prix) > 5000;
```

# Références


- Watt, Adrienne, and Nelson Eng. *Database design*. 2nd Edition. BCcampus, 2014
- Microsoft 2024, What are the SQL database functions?, accédé le 22 Novembre 2024, <https://learn.microsoft.com/en-us/sql/t-sql/functions/functions?view=sql-server-ver16\>.
- Microsoft 2024, Language Elements (Transact-SQL), accédé le 22 Novembre 2024, <https://learn.microsoft.com/en-us/sql/t-sql/language-elements/language-elements-transact-sql?view=sql-server-ver16\>.
- Microsoft 2024, Databases, accédé le 22 Novembre 2024, <https://learn.microsoft.com/en-us/sql/relational-databases/databases/databases?view=sql-server-ver16\>
- Microsoft 2024, # Transact-SQL statements, accédé le 22 Novembre 2024<https://learn.microsoft.com/en-us/sql/t-sql/statements/statements?view=sql-server-ver16\>