3. Le langage SQL DML

3.1 Petit Rappel

Comme nous avons vu dans le chapitre précédent, le sous-langage DML est une partie du langage SQL qui sert à manipuler des données et inclut la plupart des instructions SQL courantes telles que **SELECT, INSERT, UPDATE, DELETE**, etc. Il est utilisé pour **ajouter, modifier, récupérer, supprimer et mettre à jour** les données dans une base de données.

3.2 Opérateurs

Ceux-ci nous permettent de faire des calculs arithmétiques (additionner, soustraire, diviser...), logiques ou comparatifs (supérieurs à, inférieurs à...) entre colonnes, lignes, etc....

Opérateurs arithmétiques

Opérateur	Description
+	Ajouter
-	Soustraire
*	Multiplier
/	Diviser
%	Modulo

Exemple:

```
SELECT 2 + 2;
```

Opérateurs de comparaison SQL

Opérateur	Description
=	Égal à
>	Supérieur à
<	Moins que
>=	Supérieur ou égal à
<=	inférieur ou égal à
<>	N'est pas égal à

Exemples : sur une table que nous appellerons city et qui contient 2 colonnes (name et population) :

```
SELECT * FROM city WHERE population > 100000;
SELECT * FROM city WHERE population >= 100000;
SELECT * FROM city WHERE population < 100000;
```

Opérateurs logiques SQL

Opérateur	Description
ALL	VRAI si toutes les valeurs de la sous-requête remplissent la condition
AND	VRAI si toutes les conditions séparées par AND sont VRAIES
ANY	VRAI si l'une des valeurs de la sous-requête remplit la condition
BETWEEN	VRAI si l'opérande se trouve dans la plage des comparaisons
EXISTS	VRAI si la sous-requête renvoie un ou plusieurs enregistrements
IN	TRUE si l'opérande est égal à l'une d'une liste d'expressions
LIKE	VRAI si l'opérande correspond à un motif
NOT	Affiche un enregistrement si la (les) condition(s) n'est (ne sont) PAS VRAIE(S)
OR	VRAI si l'une des conditions séparées par OU est VRAIE
SOME	VRAI si l'une des valeurs de la sous-requête remplit la condition

Exemples : sur une table que nous appellerons city et qui contient 2 colonnes (name et population) :

```
SELECT * FROM city WHERE population BETWEEN 100000 AND 153106;

-- Sélectionner les villes qui commencent avec la lettre a
SELECT * FROM city WHERE name LIKE 'a%';

-- Sélectionner les villes qui finissent avec la lettre a
SELECT * FROM city WHERE name LIKE '%a';

--Sélectionner les villes commençant par "a" et d'une longueur d'au moins 3
caractères
SELECT * FROM city WHERE name LIKE 'a__%';

--Sélectionner les villes avec plus d"un million d'habitants qui ne commencent
pas par P
SELECT * FROM city WHERE population > 1000000 AND name NOT LIKE 'P%' ORDER BY
population DESC
```

Il y a d'autres opérateurs. Si vous voulez en savoir plus vous pouvez jeter un œil à la documentation de MySQL.

Instruction SELECT

L'instruction SELECT est utilisée pour **récupérer les lignes d'une ou plusieurs tables**.
SELECT peut également être utilisé pour récupérer les lignes calculées sans référence à une table.

Par exemple, prenons ce code SQL:

```
SELECT 5 - 3 ;
```

Nous allons obtenir ce résultat:

5 - 3
2

Sélectionner toutes les données

L'instruction **SELECT sans condition extrait tous les enregistrements d'une table**. L'instruction SELECT suivante récupère toutes les données de la table de livre.

```
SELECT * FROM livre;
```

Lignes spécifiques avec WHERE

Lorsqu'un utilisateur veut récupérer des lignes individuelles d'une table, une clause **WHERE** doit être ajoutée avec l'instruction **SELECT** immédiatement **suivie d'une condition**.

Exemple d'une clause WHERE permettant de retrouver les lignes de la table livre où le pays est la France:

```
SELECT * FROM  WHERE pays = "France";
```

Si notre table livre ressemble à celle-ci:

id	titre	pays	annee
1	Extension du domaine de la lutte	France	1994
3	L'Ingénieux Hidalgo Don Quichotte de la Manche	Espagne	1605
2	Stupeur et tremblements	France	1999
4	Le Livre de l'intranquillité	Portugal	1982
5	Apologie de la viande	France	1999

Nous obtiendrons le résultat suivant.

id	titre	pays	annee
1	Extension du domaine de la lutte	France	1994
2	Stupeur et tremblements	France	1999
5	Apologie de la viande	France	1999

Lignes spécifiques avec opérateur AND

L'opérateur AND est utilisé pour combiner plus d'une condition visant à récupérer des enregistrements lorsque les deux conditions sont remplies. L'instruction SELECT suivante récupère les lignes particulières où le pays et l'année de publication sont 'France' et '1999'.

```
SELECT * FROM livre WHERE pays = 'France' AND annee = 1999;
```

id	titre	pays	annee
2	Stupeur et tremblements	France	1999
5	Apologie de la viande	France	1999

Colonnes spécifiques avec SELECT

Pour récupérer les enregistrements de colonnes spécifiques, vous devez spécifier une l**iste de colonnes séparées par des virgules**. L'instruction suivante retourne le id du livre et le titre.

```
SELECT id, titre FROM livre;
```

id	titre
1	Extension du domaine de la lutte
3	L'Ingénieux Hidalgo Don Quichotte de la Manche
2	Stupeur et tremblements
4	Le Livre de l'intranquillité
5	Apologie de la viande

Colonnes spécifiques avec un opérateur distinct

La clause DISTINCT est utilisée pour récupérer des lignes uniques d'une table. L'instruction suivante récupère les années uniques de la table livre.

```
SELECT DISTINCT annee FROM livre;
```

Voici le résultat:

annee
1994
1605
1999
1982

Colonnes spécifiques avec l'opérateur logique OU

L'opérateur OU récupère les enregistrements d'une table **si au moins l'une des conditions données est satisfaite.** L'instruction suivante récupère les enregistrements des colonnes id, titre, annee de la table livre si le pays est 'Espagne' ou la colonne annee est '1999'.

```
SELECT id, titre, annee
FROM livre
WHERE pays = 'Espagne' OR annee = 1999;
```

Voici le résultat:

id	titre	annee
3	L'Ingénieux Hidalgo Don Quichotte de la Manche	1605
2	Stupeur et tremblements	1999
5	Apologie de la viande	1999

Trier les lignes par ordre croissant

La clause **ORDER BY** spécifie l'ordre dans lequel les colonnes sont triées lors de la récupération des données dans une instruction **SELECT**. **Par défaut, les colonnes sont triées par ordre croissant**, ce qui signifie que la plus petite valeur vient en premier. Vous pouvez utiliser le mot-clé ASC pour obtenir le même résultat. Pour trier dans l'**ordre inverse, la clause DESC** doit être utilisée.

Dans l'instruction suivante, tous les enregistrements des colonnes id, titre et annee de la table livre sont récupérés et triés en relation à la colonne annee. Comme nous n'avons pas spécifié de mot-clé d'ordre (ASC ou DESC), par défaut, il est trié par ordre croissant.

```
SELECT id, titre, annee FROM livre ORDER BY annee;
```

Nous obtenons le résultat suivant:

id	titre	annee
3	L'Ingénieux Hidalgo Don Quichotte de la Manche	1605
4	Le Livre de l'intranquillité	1982
1	Extension du domaine de la lutte	1994
2	Stupeur et tremblements	1999
5	Apologie de la viande	1999

Trier les lignes sur plusieurs colonnes

Le tri peut être effectué sur plusieurs colonnes. Ce type de tri s'effectue de la manière suivante : d'abord les lignes sont triées sur la première colonne, puis les lignes sont triées sur la deuxième colonne dont les données de la première colonne sont les mêmes.

Code:

```
SELECT titre ,annee, pays FROM livre ORDER BY pays, annee;
```

Résultat:

titre	annee	pays
L'Ingénieux Hidalgo Don Quichotte de la Manche	1605	Espagne
Extension du domaine de la lutte	1994	France
Stupeur et tremblements	1999	France
Apologie de la viande	1999	France
Le Livre de l'intranquillité	1982	Portugal

Sélection avec valeur NULL

IS NULL, IS NOT NULL est utilisé pour sélectionner ou **tester si une valeur stockée dans une table est NULL**. Lors de l'écriture d'une instruction, le mot-clé NULL est utilisé pour spécifier une valeur nulle.

Qu'est-ce que la valeur NULL ? :

- Une valeur de NULL indique que la valeur est inconnue, ne s'applique pas ou sera ajoutée ultérieurement.
- Une valeur de NULL n'est pas une valeur vide, une valeur égale à zéro ou une chaine de texte vide.
- Il n'y a pas deux valeurs NULL égales.
- Puisqu'une valeur de NULL est inconnue, les comparaisons entre deux valeurs nulles, ou entre une valeur NULL et toute autre valeur, retournent NULL.

Example de comparaison avec NULL:

```
SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
```

Résultat:

1 = NULL	1 <> NULL	1 < NULL	1> NULL
NULL	NULL	NULL	NULL

Nous pouvons tester d'une autre manière si la valeur est NULL avec la fonction **IS NULL**:

Code SQL :

```
SELECT 1 IS NULL, 1 IS NOT NULL;
```

Résultat :

1 IS NULL	1 IS NOT NULL
0	1

Les fonctions agrégatives (ou statistiques)

Les fonctions agrégatives récupèrent une valeur unique après avoir effectué un calcul sur un ensemble de valeurs. **En général, les fonctions agrégées ignorent les valeurs nulles.** Souvent, les fonctions globales sont accompagnées de la clause **GROUP BY** de l'instruction SELECT.

Fonction AVG()

Considérons la table livre suivante:

id	titre	prix	num_pages	pub_id
1	livre 1	19.5	250	P003
2	livre 2	20	300	P005
3	livre 3	8.44	197	P003
4	livre 4	34.99	400	P005

L'instruction suivante retournera le nombre moyen de pages (de livres) de la table livre précédente:

```
SELECT AVG(num_pages) FROM livre;
```

Voici le résultat:

AVG(num_pages)
229.4000

Fonction AVG() avec GROUP BY :

La fonction AVG() récupère la valeur moyenne d'une expression donnée pour chaque groupe si elle est utilisée avec GROUP BY.

```
SELECT pub_id, AVG(num_pages) FROM livre GROUP BY pub_id;
```

L'instruction retournera le nombre moyen de pages pour chaque groupe de la colonne 'pub_id' de la table livre.

pub_id	AVG(num_pages)
P003	223.5000
P004	350.0000

HAVING

La clause HAVING a été ajoutée à SQL parce que le mot-clé WHERE ne pouvait pas être utilisé avec les fonctions agrégatives.

Par exemple sur la table livre précédente nous pouvons exécuter l'instruction suivante:

```
SELECT COUNT(pays), pays FROM livre GROUP BY pays HAVING COUNT(pub_id) > 1;
```

Voici le résultat que nous obtenons:

COUNT(pays)	pays
3	France

Parce que dans notre table, le seul pays qui a plus d'un pub_id c'est France.

Fonction AVG() avec DISTINCT

La fonction AVG() récupère la **valeur moyenne unique** d'une expression donnée lorsqu'elle est utilisée avec le mot clé **DISTINCT**.

Fonction COUNT()

La fonction COUNT() retourne un comptage d'un nombre de valeurs qui ne sont pas NULL. Si aucune ligne n'est trouvée elle renvoie 0.

Exemple:

```
SELECT COUNT(*) FROM livre;
```

Résultat:

COUNT(*)
4

COUNT() en combinant des opérateurs logiques

L'instruction suivante retourne le nombre des livres français. La clause WHERE filtre les lignes pour le pays. Le regroupement est effectué sur les colonnes des pays par GROUPE BY, puis COUNT() compte un certain nombre d'éditeurs pour chaque groupe.

```
SELECT pays, COUNT(*)
FROM livre
WHERE pays = 'France' OR pays = 'Espagne' GROUP BY pays
```

Résultat:

pays	COUNT(*)
Espagne	1
France	3

COUNT() utilisant plusieurs tables

Considérons ces deux tables:
Table livre

id	titre	prix	num_pages	pub_id
1	livre 1	19.5	250	P003
2	livre 2	20	300	P005
3	livre 3	8.44	197	P003
4	livre 4	34.99	400	P005

Table editeur

id	name
P005	éditeur1
P003	éditeur2
P006	éditeur 3

Code:

```
SELECT editeur.name, COUNT(*)
FROM editeur,livre
WHERE editeur.id=livre.pub_id
GROUP BY editeur.id;
```

Résultat:

name	COUNT(*)
éditeur2	2
éditeur1	2

Fonction SUM()
La fonction SUM() retourne la somme d'une expression. La fonction SUM() renvoie NULL lorsque l'ensemble de données retournées n'a pas de lignes.

Prenons la table livre précédente:

```
SELECT SUM(prix)
FROM livres;
```

Résultat:

SUM(prix)
82.93

Fonction SUM() utilisant plusieurs colonnes

Imaginons que la table livres a une colonne de plus appelée quantité. Nous pourrions utiliser une instruction comme la suivante:

```
SELECT pub_id,
SUM(quatite*prix)
FROM livre
GROUP BY pub_id;
```

Fonction SUM() en combinaison avec la fonction COUNT() et utilisant des variables

L'instruction suivante retournera la somme du 'somme', une variable temporaire qui compte le nombre de livres contenant plus de 200 pages de la table 'livre'.

```
SELECT SUM(somme)
FROM(
    SELECT COUNT(*) AS somme
    FROM livre WHERE num_pages > 200
) AS bb;
```

Résultat:

SUM(somme)
3

Fonction SUM() avec clause DISTINCT

La fonction SUM() récupère la somme d'une valeur unique d'une expression si elle est accompagnée de la clause **DISTINCT**.

Imaginons que nous avons une table comme celle-ci:

id	titre	prix
1	livre 1	0
2	livre 2	0
3	livre 3	0
4	livre 4	50
5	livre 5	50

Code:

```
SELECT SUM(DISTINCT prix) FROM livre;
```

Résultat:

SUM(DISTINCT prix)
50

Par contre , si on exécute l'instruction sans la clause DISTINCT:

```
SELECT SUM(prix) FROM livre;
```

Résultat:

SUM(DISTINCT prix)
100

Bien sûr, il a un tas d'autres fonctions agrégatives comme: **MAX(), MIN() ou VARIANCE()**.

Voici la liste complète de fonctions agrégatives de MySQL: Cliquez pour voir la liste

Sous-requêtes

Une sous-requête est une requête SQL **imbriquée dans une requête**.

Imaginons une table comme celle ci que nous appellerons employe:

id	prenom	nom	salaire
1	Jean-Pierre	Duchamp	2000.00
2	Manuel	del Monte	1700.50
3	Quentin	Lasserre	1400.00
4	Harold	Binham	3000.90

Nous pourrions utiliser une sous-requête comme la suivante pour savoir s'il y a des employés qui gagnent plus que la moyenne du salaire dans notre entreprise.

```
SELECT id, prenom, nom, salaire
       FROM employe WHERE salaire >
            (SELECT AVG(salaire) FROM employe);
```

Souvent, les fonctions globales sont accompagnées de la clause **GROUP BY** de l'instruction SELECT.