Table des matières SQL Syntaxe

SQL SYNTAXE

I. SELECT.

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans une table de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table. Pour cela, il suffit tout simplement de séparer les noms des colonnes souhaitées par une virgule.

```
SELECT attribut1, attribut2, attribut3 FROM nom_table;
```

Pour obtenir toutes les colonnes d'une table :

```
SELECT * FROM nom_table;
```

II. DISTINCT.

L'utilisation de la commande SELECT permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande 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 attribut FROM nom_table;
```

L'utilisation de la commande DISTINCT est très pratique pour éviter les résultats en doubles. Cependant, pour optimiser les performances il est préférable d'utiliser la commande SQL GROUP BY lorsque c'est possible.

III. 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.

```
SELECT * FROM nom_table WHERE condition;
```

Il existe plusieurs opérateurs de comparaisons. Voir l'aide mémoire pour les opérateurs les plus couramment utilisés.

IV. AND et OR.

Une requête SQL peut être restreinte à l'aide de la condition WHERE. Les opérateurs logiques AND et OR peuvent être utilisées au sein de la commande WHERE pour combiner des conditions.

Les opérateurs sont à ajoutés dans la condition WHERE. Ils peuvent être combinés à l'infini pour filtrer les données comme souhaités.

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

```
SELECT attribut(s)
FROM nom_table
WHERE condition1 AND condition2;
```

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

```
SELECT attribut(s) FROM nom_table
WHERE condition1 OR condition2;
```

Ces opérateurs peuvent être combinés à l'infini et mélangés. L'exemple ci-dessous filtre les résultats de la table "nom_table" si condition1 **ET** (condition2 **OU** condition3) est vrai :

```
SELECT attribut(s) FROM nom_table
WHERE condition1 AND (condition2 OR condition3);
```

Attention : il faut penser à utiliser des parenthèses lorsque c'est nécessaire. Cela permet d'éviter les erreurs car et ça améliore la lecture d'une requête par un humain.

V. IN.

L'opérateur logique IN dans SQL s'utilise avec la commande WHERE pour vérifier si une colonne est égale à une des valeurs comprise dans set 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 "attribut" est égale à 'valeur 1' OU 'valeur 2' ou 'valeur 3', il est possible d'utiliser la syntaxe suivante.

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

A savoir : entre les parenthèses il n'y a pas de limite du nombre d'arguments. Il est possible d'ajouter encore d'autres valeurs.

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

VI. BETWEEN.

L'opérateur BETWEEN est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant WHERE. L'intervalle peut être constitué de chaînes de caractères, de nombres ou de dates. L'exemple le plus concret consiste par exemple à récupérer uniquement les enregistrements entre 2 dates définies.

La requête suivante retournera toutes les lignes dont la valeur de la colonne "attribut" sera comprise entre valeur1 et valeur2.

```
SELECT *
FROM nom_table
WHERE attribut BETWEEN 'valeur1' AND 'valeur2';
```

Toutes les bases de données ne gèrent pas l'opérateur BETWEEN de la même manière. Certains systèmes vont inclurent les valeurs qui définissent l'intervalle tandis que d'autres systèmes considèrent ces valeurs sont exclues.

VII. LIKE.

L'opérateur LIKE est utilisé dans la clause WHERE des requêtes SQL. Ce mot-clé 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 multiples.

```
SELECT *
FROM nom_table
WHERE attribut LIKE modele;
```

- LIKE '%a' : le caractère '%' est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaines de caractère qui se termine par un "a".
- LIKE 'a%' : ce modèle permet de rechercher toutes les lignes de "colonne" qui commence par un "a".
- LIKE '%a%' : ce modèle est utilisé pour rechercher tous les enregistrement qui utilisent le caractère "a".
- LIKE 'pa%on' : ce modèle permet de rechercher les chaines qui commence 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".

VIII. IS NULL / IS NOT NULL.

L'opérateur IS permet de filtrer les résultats qui contiennent la valeur NULL. Cet opérateur est indispensable car la valeur NULL est une valeur inconnue et ne peut par conséquent pas être filtrée par les opérateurs de comparaison (cf. égal, inférieur, supérieur ou différent).

```
SELECT *
FROM nom_table
WHERE attribut IS NULL;
```

A l'inverse, pour filtrer les résultats et obtenir uniquement les enregistrements qui ne sont pas null.

```
SELECT *
FROM nom_table
WHERE attribut IS NOT NULL;
```

L'opérateur IS retourne en réalité un booléen. Cet opérateur est souvent utilisé avec la condition WHERE mais peut aussi trouvé son utilité lorsqu'une sous-requête est utilisée.

IX. GROUP BY.

La commande GROUP BY est utilisée pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat.

```
SELECT attribut1, fonction(attribut2)
FROM nom_table
GROUP BY attribut1;
```

Cette commande doit toujours s'utiliser après la commande WHERE et avant la commande HAVING.

La manière simple de comprendre le GROUP BY c'est tout simplement d'assimiler qu'il va éviter de présenter plusieurs fois les mêmes lignes. C'est une méthode pour éviter les doublons.

Les fonctions les plus courantes sont :

- **AVG()** : permet de calculer la moyenne de plusieurs valeurs ;
- **COUNT()** : permet de compter le nombre de lignes concernées ;
- **MAX()** : permet de récupérer la valeur maximale ;
- **MIN()** : permet de récupérer la valeur minimale ;
- **SUM( )** : permet d'additionner les valeurs de plusieurs lignes ;

X. HAVING.

La condition HAVING est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que **SUM()**, **COUNT()**, **AVG()**, **MIN()** ou **MAX()**.

```
SELECT attribut1, SUM(attribut2)
FROM nom_table
GROUP BY attribut1
HAVING fonction(attribut2) operateur valeur;
```

Cela permet donc de sélectionner les colonnes de la table "nom_table" en GROUPANT les lignes qui ont des valeurs identiques sur la colonne "attribut1" et que la condition de HAVING soit respectée.

Important : HAVING est très souvent utilisé en même temps que GROUP BY mais ce n'est pas obligatoire.

XI. ORDER BY.

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

```
SELECT attribut1, attribut2
FROM nom_table
ORDER BY attribut1;
```

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

```
SELECT attribut1, attribut2, attribut3
FROM nom_table
ORDER BY attribut1 DESC,  attribut2 ASC;
```

Pour obtenir la liste des tomates par masse croissante :

```
SELECT *
FROM tomate
ORDER BY masse;
```

XII. AS.

Il est possible d'utiliser des alias pour renommer temporairement une colonne ou une table dans une requête. Cette astuce est particulièrement utile pour faciliter la lecture des requêtes.

12.1 Alias sur une colonne.

Un alias permet de renommer le nom d'une colonne dans les résultats d'une requête SQL. C'est pratique pour avoir un nom facilement identifiable dans une application qui doit ensuite exploiter les résultats d'une recherche.

```
SELECT attribut1 AS nouveau_attribut1
FROM nom_table;
```

Pour obtenir la liste des masses moyennes des tomates selon leur couleur et dont la masse moyenne par couleur est supérieure à 100 g :

```
SELECT couleur, AVG(masse) AS Masse_moy
FROM tomate
GROUP BY couleur
HAVING AVG(masse) > 100;
```

12.2 Alias sur une table.

Permet d'attribuer un autre nom à une table dans une requête SQL. Cela peut aider à avoir des noms plus court, plus simple et plus facilement compréhensible. Ceci est particulièrement vrai lorsqu'il y a des jointures.

```
SELECT attribut1, attribut2
FROM nom_table AS nouveau_nom_table;
```

XIII. Fonctions d'agrégation.

L'agrégation consiste à regrouper tous les tuples d'une table ayant même valeur pour un ou plusieurs attributs. Dans le résultat n'apparaîtra alors que le dernier de ces tuples, il faut imaginer que tous les autres sont cachés derrière.
On a généralement besoin de sélectionner certains tuples de la table :

- si on sélectionne avant de regrouper, on utilise WHERE;
- si on sélectionne après le regroupement, on utilise HAVING.

Syntaxe de l'agrégation :

```
SELECT attributs
FROM nom_table;
(WHERE condition préliminaire)
GROUP BY attribut_1, …, attribut_n
(HAVING condition finale)
```

- Les 3 premières lignes créent une table ;
- la suivante effectue l'agrégation dans cette table : regroupement des tuples ayant même valeur sur chacune des colonnes attribut_1, ..., attribut_n ;
- la dernière ligne ne garde, dans cette table regroupée, que les lignes vérifiant la condition finale.

L’utilisation la plus basique consiste à utiliser la syntaxe suivante :

```
SELECT fonction(attribut) FROM nom_table;
```

13.1 AVG().

La fonction d’agrégation **AVG()** permet de calculer une valeur moyenne sur un ensemble d’enregistrement de type numérique et non nul.

```
SELECT AVG(attribut) FROM nom_table;
```

Cette requête permet de calculer la valeur moyenne de la colonne "attribut" sur tous les enregistrements de la table "nom_table". Il est possible de filtrer les enregistrements concernés à l’aide de la commande WHERE. Il est aussi possible d’utiliser la commande GROUP BY pour regrouper les données appartenant à la même entité.

13.2 COUNT().

Si on souhaite connaître le **nombre d'enregistrement** (les valeurs NULL ne sont pas comptabilisées) d'une colonne.

```
SELECT COUNT(attribut) FROM nom_table;
```

Si on souhaite connaître le **nombre total de lignes** (les valeurs NULL sont comptabilisées) de la table.

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

Enfin, il est également possible de compter le nombre d’enregistrement distinct pour une colonne. La fonction ne comptabilisera ni les doublons, ni les valeurs NULL pour une colonne choisie.

```
SELECT COUNT(DISTINCT attribut) FROM nom_table;
```

en général, en terme de performance il est plutôt conseillé de filtrer les lignes avec GROUP BY si c’est possible, puis d’effectuer un **COUNT**(*).

13.3 MAX().

La fonction d’agrégation **MAX()** permet de retourner la valeur maximale d’une colonne dans un set d’enregistrement. La fonction peut s’appliquée à des données numériques ou alphanumériques.

```
SELECT MAX(attribut) FROM nom_table;
```

Lorsque cette fonctionnalité est utilisée en association avec la commande GROUP BY, la requête peut ressembler à l’exemple ci-dessous.

```
SELECT attribut1, MAX(attribut2)
FROM nom_table
GROUP BY attribut1;
```

13.4 MIN().

La fonction d’agrégation **MIN()** permet de retourner la valeur minimale d’une colonne dans un set d’enregistrement. La fonction peut s’appliquée à des données numériques ou alphanumériques.

```
SELECT MIN(attribut) FROM nom_table;
```

Lorsque cette fonctionnalité est utilisée en association avec la commande GROUP BY, la requête peut ressembler à l'exemple ci-dessous.

```
SELECT attribut1, MIN(attribut2)
FROM nom_table
GROUP BY attribut1;
```

13.5 SUM().

La fonction d'agrégation **SUM()** permet de calculer la somme totale d'une colonne contenant des valeurs numériques. Cette fonction ne fonctionne que sur des colonnes de types numériques (INT, FLOAT …) et n'additionne pas les valeurs NULL.

```
SELECT SUM(attribut) FROM nom_table;
```

Il est possible de filtrer les enregistrements avec la commande WHERE pour ne calculer la somme que des éléments souhaités.

```
SELECT SUM(attribut2)
FROM nom_table
WHERE condition;
```

Il est possible de calculer la somme de chaque paramètre de la colonne 1 en groupant les lignes avec la commande GROUP BY.

```
SELECT attribut1, SUM(attribut2)
FROM nom_table
GROUP BY attribut1;
```

XIV. Conclusion.

L'agrégation consiste à regrouper tous les tuples d'une table ayant même valeur pour un ou plusieurs attributs. Dans le résultat n'apparaîtra alors que le dernier de ces tuples, il faut imaginer que tous les autres sont cachés derrière.

On a généralement besoin de sélectionner certains tuples de la table :

- si on sélectionne avant de regrouper, on utilise WHERE;
- si on sélectionne après le regroupement, on utilise HAVING.

Une requête SELECT peut devenir assez longue. Juste à titre informatif, voici une requête SELECT qui possède presque toutes les commandes possibles.

```
1  SELECT *
2  FROM nom_table
3  WHERE condition
4  GROUP BY expression
5  HAVING condition
   { UNION | INTERSECT | EXCEPT }
6  ORDER BY expression
7  LIMIT count
8  OFFSET start
```

- Les trois premières lignes créent une table ;
- La quatrième ligne effectue l'agrégation dans cette table : regroupement des tuples ayant même valeur sur chacune des colonnes attribut1, ..., attributn.
- La cinquième ligne ne garde, dans cette table regroupée, que les lignes vérifiant la condition initiale.
- La sixième ligne trie les lignes par ordre croissant du paramètre expression.
- La septième limite le nombre de lignes renvoyées.
- La dernière ligne permet d'effectuer un décalage sur les résultats.