



LA MINUTE DE CODE

MySQL™

SQL

PARTIE 3

FONCTIONS D'AGRÉGATION | FONCTIONS MATHÉMATIQUES |
OPÉRATEURS DE COMPARAISON | REQUÊTES AVANCÉES | REQUÊTES IMBRIQUÉS

- 
- Fonctions d'agrégation
 - Fonctions mathématiques
 - Opérateurs de comparaison
 - Requêtes avancées
 - Requêtes imbriqués

La fonction AVG()

La fonction AVG en SQL (Structured Query Language) est utilisée pour calculer la moyenne des valeurs d'une colonne numérique dans une table de base de données.

La syntaxe générale de la fonction AVG est la suivante :

```
SELECT AVG(column_name) FROM table_name;
```

où column_name est le nom de la colonne numérique pour laquelle vous souhaitez calculer la moyenne, et table_name est le nom de la table qui contient cette colonne.

Par exemple, si vous avez une table employes avec une colonne score qui contient les scores de vente, vous pouvez utiliser la fonction AVG pour calculer la moyenne des scores de tous les employés en utilisant la requête suivante :

```
SELECT AVG(score) FROM employes;
```

La fonction AVG renverra une seule valeur, qui est la moyenne des scores de tous les employés dans la table.

Si la colonne contient des valeurs NULL, la fonction AVG les ignore et calcule la moyenne uniquement pour les valeurs non NULL.

La fonction count()

La fonction COUNT en SQL (Structured Query Language) est utilisée pour compter le nombre de lignes dans une table ou le nombre de valeurs non NULL dans une colonne de cette table.

La syntaxe générale de la fonction COUNT est la suivante :

```
SELECT COUNT(column_name) FROM table_name;
```

ou

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

La première requête compte le nombre de valeurs non NULL dans une colonne spécifique, tandis que la seconde requête compte le nombre total de lignes dans la table, indépendamment de la colonne.

Par exemple, si vous avez une table employes avec une colonne id qui contient les identifiants des étudiants, vous pouvez utiliser la fonction COUNT pour compter le nombre total d'employés en utilisant la requête suivante :

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

La fonction COUNT renverra une seule valeur, qui est le nombre total de lignes dans la table. Si vous souhaitez compter uniquement le nombre d'employés ayant un score (par exemple pour le nombre de vente) non NULL, vous pouvez utiliser la requête suivante :

```
SELECT COUNT(score) FROM employes;
```

La fonction COUNT renverra le nombre d'employés ayant un score non NULL dans la colonne score. Si la colonne ne contient que des valeurs NULL, la fonction COUNT renverra 0.

La fonction max()

La fonction MAX en SQL (Structured Query Language) est utilisée pour obtenir la valeur maximale d'une colonne numérique dans une table de base de données.

La syntaxe générale de la fonction MAX est la suivante :

```
SELECT MAX(column_name) FROM table_name;
```

où column_name est le nom de la colonne numérique pour laquelle vous souhaitez trouver la valeur maximale, et table_name est le nom de la table qui contient cette colonne.

Par exemple, si vous avez une table employes avec une colonne score qui contient les scores des employés, vous pouvez utiliser la fonction MAX pour obtenir le score maximal de tous les employés en utilisant la requête suivante :

```
SELECT MAX(score) FROM employes;
```

La fonction MAX renverra une seule valeur, qui est le score maximal de tous les employés dans la table. Si la colonne contient des valeurs NULL, la fonction MAX renverra la valeur NULL.

Il est important de noter que la fonction MAX peut également être utilisée avec des colonnes non numériques telles que les dates et les chaînes de caractères. Dans ce cas, la fonction MAX renverra la valeur la plus élevée en fonction de l'ordre de tri défini pour le type de données de la colonne. Par exemple, pour une colonne de dates, la fonction MAX renverra la date la plus récente.

La fonction min()

La fonction MIN en SQL (Structured Query Language) est utilisée pour obtenir la valeur minimale d'une colonne numérique dans une table de base de données.

La syntaxe générale de la fonction MIN est la suivante :

```
SELECT MIN(column_name) FROM table_name;
```

où column_name est le nom de la colonne numérique pour laquelle vous souhaitez trouver la valeur minimale, et table_name est le nom de la table qui contient cette colonne.

Par exemple, si vous avez une table employes avec une colonne score qui contient les scores des employés, vous pouvez utiliser la fonction MIN pour obtenir le score minimal de tous les employés en utilisant la requête suivante :

```
SELECT MIN(score) FROM employes;
```

La fonction MIN renverra une seule valeur, qui est le score minimal de tous les employés dans la table. Si la colonne contient des valeurs NULL, la fonction MIN renverra la valeur NULL.

Il est important de noter que la fonction MIN peut également être utilisée avec des colonnes non numériques telles que les dates et les chaînes de caractères. Dans ce cas, la fonction MIN renverra la valeur la plus basse en fonction de l'ordre de tri défini pour le type de données de la colonne. Par exemple, pour une colonne de dates, la fonction MIN renverra la date la plus ancienne.

La fonction sum()

La fonction SUM en SQL (Structured Query Language) est utilisée pour obtenir la somme des valeurs d'une colonne numérique dans une table de base de données.

La syntaxe générale de la fonction SUM est la suivante :

```
SELECT SUM(column_name) FROM table_name;
```

où column_name est le nom de la colonne numérique pour laquelle vous souhaitez trouver la somme, et table_name est le nom de la table qui contient cette colonne.

Par exemple, si vous avez une table avec une colonne score qui contient les scores, vous pouvez utiliser la fonction SUM pour obtenir le montant total des scores en utilisant la requête suivante :

```
SELECT SUM(score) FROM employes;
```

La fonction SUM renverra une seule valeur, qui est la somme des montants de scores dans la table. Si la colonne contient des valeurs NULL, la fonction SUM les ignore et calcule la somme uniquement pour les valeurs non NULL.

Il est important de noter que la fonction SUM ne fonctionne qu'avec des colonnes numériques, et ne peut pas être utilisée avec des colonnes contenant des valeurs non numériques telles que des dates ou des chaînes de caractères.

La fonction round()

La fonction ROUND en SQL (Structured Query Language) est utilisée pour arrondir un nombre décimal à un certain nombre de décimales spécifié.

La syntaxe générale de la fonction ROUND est la suivante :

```
SELECT ROUND(colonne) FROM table;
```

où numeric_expression est l'expression numérique à arrondir et length est le nombre de décimales à conserver. Si length est omis, la fonction ROUND arrondira numeric_expression à un nombre entier.

Par exemple, si vous avez un nombre décimal x et que vous souhaitez l'arrondir à deux décimales, vous pouvez utiliser la fonction ROUND de la manière suivante :

```
SELECT ROUND(score) AS nouveau_score FROM employes WHERE emp_id = 3;
```

La fonction ROUND renverra le nombre x arrondi à deux décimales. Il est important de noter que la fonction ROUND arrondit les nombres en fonction de la règle de l'arrondi mathématique standard.

Si la valeur à arrondir est exactement à mi-chemin entre deux valeurs possibles, la fonction ROUND arrondit à l'une des deux valeurs en fonction de l'arrondi "pair-impair".

Cela signifie que si le chiffre suivant le dernier chiffre à conserver est impair, le dernier chiffre sera arrondi vers le haut. Si le chiffre suivant le dernier chiffre à conserver est pair, le dernier chiffre sera arrondi vers le bas.

Si je veut arrondir un chiffre après la virgule je rajoute le paramètre 1:

```
SELECT ROUND(score, 1) AS nouveau_score  
FROM employes WHERE emp_id = 3;
```

La fonction ceil()

La fonction CEILING est utilisée pour arrondir un nombre décimal à l'entier supérieur le plus proche.

La syntaxe générale de la fonction CEILING est la suivante :

```
SELECT CEIL(colonne) FROM table;
```

où numeric_expression est le nombre décimal à arrondir.

Par exemple, si vous avez un nombre décimal x et que vous souhaitez l'arrondir à l'entier supérieur le plus proche, vous pouvez utiliser la fonction CEILING de la manière suivante :

```
SELECT CEIL(score) FROM employes WHERE  
emp_id = 2;
```

La fonction CEILING renverra le plus petit nombre entier supérieur ou égal à x .

Il est important de noter que la fonction CEILING arrondit toujours vers le haut, même si le nombre décimal est déjà un entier. Si le nombre décimal est négatif, la fonction CEILING renverra le plus grand nombre entier inférieur ou égal à x .

La fonction floor()

La fonction FLOOR en SQL (Structured Query Language) est utilisée pour arrondir un nombre décimal à l'entier inférieur le plus proche.

La syntaxe générale de la fonction FLOOR est la suivante :

```
SELECT FLOOR(colonne) FROM table;
```

où numeric_expression est le nombre décimal à arrondir.

Par exemple, si vous avez un nombre décimal x et que vous souhaitez l'arrondir à l'entier inférieur le plus proche, vous pouvez utiliser la fonction FLOOR de la manière suivante :

```
SELECT FLOOR(score) FROM employes WHERE  
emp_id = 2;
```

La fonction FLOOR renverra le plus grand nombre entier inférieur ou égal à x.

Il est important de noter que la fonction FLOOR arrondit toujours vers le bas, même si le nombre décimal est déjà un entier. Si le nombre décimal est négatif, la fonction FLOOR renverra le plus petit nombre entier supérieur ou égal à x.

En SQL, l'opérateur de comparaison est utilisé pour comparer deux valeurs et renvoyer un résultat vrai ou faux (c'est-à-dire une valeur booléenne).

```
SELECT colonne FROM table WHERE colonne opérateur condition
```

Les opérateurs de comparaison couramment utilisés en SQL sont :

- "!=" : renvoie vrai si les deux valeurs sont égales, faux sinon.
- "<>" ou "!=" : renvoie vrai si les deux valeurs sont différentes, faux sinon.
- "<" : renvoie vrai si la première valeur est inférieure à la deuxième valeur, faux sinon.
- ">" : renvoie vrai si la première valeur est supérieure à la deuxième valeur, faux sinon.
- "<=" : renvoie vrai si la première valeur est inférieure ou égale à la deuxième valeur, faux sinon.
- ">=" : renvoie vrai si la première valeur est supérieure ou égale à la deuxième valeur, faux sinon.

Voici quelques exemples d'utilisation des opérateurs de comparaison en SQL :

```
SELECT * FROM employes WHERE score > 500
```

```
SELECT * FROM employes WHERE score < 500
```

```
SELECT * FROM employes WHERE score <= 500
```

```
SELECT * FROM employes WHERE score >= 500
```

```
SELECT * FROM employes WHERE score = 500
```

Groupe by

En SQL, la clause GROUP BY est utilisée pour regrouper les résultats d'une requête en fonction des valeurs d'une ou plusieurs colonnes. Elle est souvent utilisée en conjonction avec des fonctions d'agrégation telles que COUNT, SUM, AVG, MAX, et MIN pour calculer des statistiques sur les groupes de données.

La syntaxe est la suivante:

```
SELECT function d'agrégation(colonne) FROM  
table GROUP BY colonne
```

Exemple:

```
SELECT prod_cat, AVG(prod_prix) AS  
prix_moyen, COUNT(prod_id) AS prod_nb,  
MAX(prod_prix) AS prod_max, MIN(prod_prix)  
AS prod_min FROM products GROUP BY  
prod_cat;
```

Between

La clause BETWEEN en SQL est une condition qui permet de sélectionner des données dans une plage de valeurs spécifiée. Elle est utilisée dans la clause WHERE pour filtrer les résultats d'une requête.

La syntaxe de la clause BETWEEN est la suivante:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND
value2;
```

Exemple:

```
SELECT prod_nom, prod_prix
FROM products
WHERE prod_prix BETWEEN 50 AND 400;
```

Exemple avec des dates

```
SELECT prod_nom, prod_prix
FROM products
WHERE prod_date BETWEEN '2017-05-02' AND
'2020-05-26';
```

Having

La clause HAVING en SQL est utilisée avec la clause GROUP BY pour filtrer les résultats de la requête qui contiennent des groupes de données agrégées. La clause HAVING permet de spécifier une condition pour les groupes de résultats retournés.

La syntaxe de la clause Having est la suivante:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition;
```

Exemples:

```
SELECT shop_id, SUM(vente) as total
FROM employes
GROUP BY shop_id
HAVING SUM(vente) > 500;
```

Ceci va m'afficher les magasin qui ont un score de vente supérieur à 500

```
SELECT prod_cat, AVG(prod_prix) AS
prix_moyen FROM products GROUP BY prod_cat
```

Ceci va m'afficher les catégories avec le prix moyens des produits inférieures à 100

Like

La commande LIKE est utilisée en SQL pour rechercher des motifs (patterns) dans une colonne de texte.

La commande ILIKE (ou LIKE insensible à la casse, selon les systèmes de gestion de bases de données) est similaire à LIKE, mais elle permet de faire des recherches en ignorant la casse (majuscules et minuscules).

La syntaxe de la commande ILIKE est la suivante:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;
```

Dans cette syntaxe, "column_name(s)" est le nom de la ou des colonnes dans lesquelles vous voulez rechercher le motif, "table_name" est le nom de la table dans laquelle vous voulez effectuer la recherche, "pattern" est le motif que vous voulez rechercher.

Le motif peut contenir des caractères spéciaux pour effectuer des recherches plus avancées, comme:

- % : Représente 0, 1 ou plusieurs caractères (joker).
- _ : Représente un seul caractère (joker).
- [] : Représente un ensemble de caractères.
- [^] : Représente tout caractère qui n'est pas dans l'ensemble spécifié.

Voici un exemple de recherche en utilisant la commande ILIKE :

La fonction concat

La fonction CONCAT en SQL est utilisée pour concaténer ou joindre deux ou plusieurs chaînes de caractères en une seule chaîne. Elle prend en entrée deux ou plusieurs arguments, qui peuvent être des colonnes, des variables ou des chaînes littérales.

La syntaxe générale de la fonction CONCAT est la suivante :

```
SELECT CONCAT(colonne, ' ', colonne) AS  
alias  
FROM table;
```

Exemple 1:

```
SELECT CONCAT(emp_nom, emp_prenom) AS  
nouveau  
FROM employes;
```

Exemple 2:

Il m'affichera le nom avec le prenom

```
SELECT CONCAT('Nom:', emp_nom, ' Prenom:  
' , emp_prenom) AS nouveau  
FROM employes;
```

Il m'affichera le nom avec le prenom + les textes rajoutés "nom" et "prenom"

Les sous requêtes SELECT

```
SELECT colonne, WHERE colonne OPERATEUR  
(sous-requête) FROM table
```

En SQL, une sous-requête (ou sous-query) est une requête imbriquée à l'intérieur d'une requête principale. Les sous-requêtes sont souvent utilisées pour effectuer des calculs ou des filtrages complexes en utilisant les données d'une table ou d'une vue dans une autre table.

L'une des utilisations courantes des sous-requêtes est de filtrer les résultats d'une requête principale. Une sous-requête est dite requête externe tandis qu'une requête principale est dite requête interne.

Par exemple, supposons que vous ayez une table "employes" avec une colonne "vente", et vous souhaitez sélectionner tous les employés dont leur vente est supérieur à la vente moyenne. Vous pouvez utiliser une sous-requête pour calculer la moyenne des ventes, puis utiliser cette valeur pour filtrer les résultats de la requête principale :

```
SELECT *  
FROM employes  
WHERE vente > (SELECT AVG(vente) FROM  
employes)
```

Exemple 2: afficher le prix le moins cher de chaque produits de chaque catégorie avec une sous requête SELECT:

```
SELECT T1.produit_cat,
MIN(T1.produit_prix) AS prixMoinsCher,
(SELECT T2.produit_nom FROM produits T2
where T2.produit_prix =
MIN(T1.produit_prix)) AS nomProduit FROM
produits T1
GROUP BY T1.produit_cat
```

Ce code SQL effectue une requête de sélection sur une table nommée "produits" et renvoie trois colonnes de données agrégées pour chaque catégorie de produits distincte présente dans la table.

Plus précisément, la requête sélectionne les colonnes "produit_cat" et "produit_prix" de la table "produits" (en utilisant l'alias "T1" pour la table), et pour chaque catégorie de produits distincte dans la table, elle retourne le prix minimum pour cette catégorie (en utilisant la fonction d'agrégation MIN) sous le nom "prixMoinsCher".

La troisième colonne, "nomProduit", est dérivée d'une sous-requête qui sélectionne le nom du produit ayant le prix minimum pour chaque catégorie de produits distincte. La sous-requête utilise à nouveau la table "produits" (en utilisant l'alias "T2" pour la table) et compare le prix minimum pour chaque catégorie de produits distincte avec le prix du produit correspondant. Le nom du produit ayant le prix minimum pour chaque catégorie de produits est renvoyé dans la colonne "nomProduit".

Enfin, la requête groupe les résultats par catégorie de produits distincte en utilisant la clause GROUP BY pour renvoyer une seule ligne pour chaque catégorie de produits distincte présente dans la table "produits".

Les sous requêtes FROM

```
SELECT colonne FROM (sous-requête) AS  
alias
```

Une sous-requête FROM (ou sous-requête dans la clause FROM) est une requête SQL imbriquée dans la clause FROM d'une requête principale.

Cette requête permet de sélectionner une table temporaire pour être utilisée comme source de données pour la requête principale.

La sous-requête FROM est souvent utilisée pour agréger des données à partir de plusieurs tables, ou pour effectuer des jointures complexes. Voici un exemple de sous-requête FROM pour afficher la quantité moyenne des deux catégories de produits :

```
SELECT AVG(total)  
FROM(  
SELECT SUM(produit_quant) AS total FROM  
produits GROUP BY produit_cat) as  
totalGlobal
```

Les sous requêtes dans les conditions

```
SELECT colonne FROM table WHERE colonne =  
(sous-requête)
```

Les sous-requêtes (ou sous-queries) dans les conditions en SQL sont des requêtes imbriquées dans une requête plus large qui permettent d'effectuer une opération de filtrage plus complexe.

Par exemple pour afficher le nom du produit le plus cher:

```
SELECT T1.produit_nom FROM produits T1  
WHERE T1.produit_prix = (SELECT  
MAX(T2.produit_prix) from produits T2)
```

1. Réaliser une sous requête dans une condition pour afficher le nom du produit et la quantité en stock du produit qui possède le plus de produits en stock.
2. Réaliser une sous-requête from pour afficher la moyenne du total des prix de tous les produits de chaque catégories.
3. Réaliser une commande pour concatainer le nom des produits et leur quantité avec un espace entre.
4. Réaliser une commande pour afficher le prix du produit le plus cher.
5. Fait de même que la consigne précédente avec le produit le moins cher

Exemple 1:

```
SELECT *
FROM employes
WHERE emp_nom LIKE 'Jo%' ;
```

Il m'affichera tout les noms qui commence par les lettre jo dans ma table employes.

Exemple 2:

```
SELECT *
FROM produits
WHERE produit_nom LIKE '%n' ;
```

Il m'affichera tout les produits qui termine par les lettre n dans ma table produits.



laminutedecode.io

@LAMINUTEDECODE

