Module 3 : Filtrage des données avec WHERE

Objectifs pédagogiques

À la fin de ce module, vous serez capable de :

- Utiliser la clause WHERE pour filtrer les résultats de requêtes
- Maîtriser les différents opérateurs de comparaison
- Combiner plusieurs conditions avec les opérateurs logiques
- Utiliser l'opérateur LIKE pour les recherches de motifs
- Gérer correctement les valeurs NULL dans vos filtres

1. Introduction à la clause WHERE

1.1 Rôle et importance du filtrage

La clause WHERE est essentielle pour extraire uniquement les données pertinentes d'une table. Sans elle, les requêtes retourneraient systématiquement toutes les lignes, ce qui serait :

- Inefficace en termes de performances
- Peu pratique pour l'analyse des données
- Inadapté aux besoins métier spécifiques

La clause WHERE agit comme un filtre qui évalue chaque ligne de la table et ne retourne que celles qui satisfont la condition spécifiée.

1.2 Syntaxe de base

La structure générale d'une requête SELECT avec WHERE est :

```
SELECT colonne1, colonne2, ...  
FROM table  
WHERE condition;
```

La condition après WHERE est une expression booléenne qui est évaluée pour chaque ligne de la table. Seules les lignes pour lesquelles cette expression est VRAIE seront incluses dans les résultats.

1.3 Position dans la requête SQL

La clause WHERE vient toujours après la clause FROM et avant d'autres clauses comme ORDER BY ou LIMIT :

```
SELECT colonne1, colonne2  
FROM table  
WHERE condition  
ORDER BY colonne1  
LIMIT 10;
```

2. Opérateurs de comparaison

2.1 Opérateurs de base

SQL fournit plusieurs opérateurs pour comparer des valeurs :

Opérateur	Description	Exemple	SQL	Signification	WHERE
=	Égal à	WHERE prix = 100	=	Égal à	WHERE prix = 100
<> ou !=	Différent de	WHERE prix <> 100	<> ou !=	Différent de	WHERE prix <> 100
<	Inférieur à	WHERE prix < 100	<	Inférieur à	WHERE prix < 100
>	Supérieur à	WHERE prix > 100	>	Supérieur à	WHERE prix > 100
<=	Inférieur ou égal à	WHERE prix <= 100	<=	Inférieur ou égal à	WHERE prix <= 100
>=	Supérieur ou égal à	WHERE prix >= 100	>=	Supérieur ou égal à	WHERE prix >= 100

2.2 Exemples d'utilisation

```
-- Trouver les produits dont le prix est supérieur à 50 €
```

```
SELECT nom_produit, prix  
FROM Produits  
WHERE prix > 50;
```

```
-- Trouver les clients qui habitent à Paris
```

```
SELECT nom, prenom  
FROM Clients  
WHERE ville = 'Paris';
```

```
-- Trouver les commandes passées avant une certaine date
```

```
SELECT commande_id, date_commande  
FROM Commandes  
WHERE date_commande < '2023-01-01';
```

2.3 Comparaison de différents types de données

Les comparaisons fonctionnent sur différents types de données, mais avec quelques particularités :

Chaînes de caractères : La comparaison dépend souvent de la collation (sensibilité à la casse, aux accents, etc.)

```
-- Selon la collation, peut être sensible ou non à la casse
```

```
SELECT * FROM Clients WHERE ville = 'paris'; -- Peut ou non correspondre à 'Paris'
```

Dates : Les formats peuvent varier selon le SGBD, mais ISO 'YYYY-MM-DD' est généralement accepté

```
SELECT * FROM Commandes WHERE date_commande = '2023-01-15';
```

Nombres : La comparaison numérique standard s'applique

```
SELECT * FROM Produits WHERE prix * 1.2 > 100; -- Le calcul est effectué avant la comparaison
```

3. Opérateurs logiques

3.1 AND, OR et NOT

Les opérateurs logiques permettent de combiner plusieurs conditions :

Opérateur	Description	Exemple
AND	Toutes les conditions doivent être vraies	WHERE prix > 50 AND stock > 0
OR	Au moins une condition doit être vraie	WHERE ville = 'Paris' OR ville = 'Lyon'
NOT	Inverse la condition	WHERE NOT ville = 'Paris'

3.2 Combinaison d'opérateurs logiques

Vous pouvez combiner plusieurs opérateurs logiques dans une même clause WHERE :

```
-- Produits entre 50 et 100 €
SELECT nom_produit, prix
FROM Produits
WHERE prix >= 50 AND prix <= 100;
```

```
-- Clients de Paris ou Lyon qui se sont inscrits en 2022
SELECT nom, prenom, ville
FROM Clients
WHERE (ville = 'Paris' OR ville = 'Lyon') AND date_inscription >= '2022-01-01' AND date_inscription <= '2022-12-31';
```

3.3 Précédence des opérateurs

Par défaut, AND a une priorité plus élevée que OR. Utilisez des parenthèses pour contrôler l'ordre d'évaluation :

```
-- Sans parenthèses : l'AND est évalué en premier
SELECT * FROM Produits WHERE categorie_id = 1 OR categorie_id = 2 AND prix < 50;
-- Équivalent à : WHERE categorie_id = 1 OR (categorie_id = 2 AND prix < 50)

-- Avec parenthèses pour changer l'ordre d'évaluation
SELECT * FROM Produits WHERE (categorie_id = 1 OR categorie_id = 2) AND prix < 50;
```

4. Recherche de motifs avec LIKE

4.1 Utilisation de l'opérateur LIKE

L'opérateur LIKE permet de rechercher des motifs dans les chaînes de caractères :

```
SELECT nom_produit
FROM Produits
```

```
WHERE nom_produit LIKE 'Smartphone%';
```

4.2 Caractères spéciaux (jokers)

Deux caractères spéciaux sont utilisés avec LIKE :

- `%` : Correspond à zéro, un ou plusieurs caractères
- `_` : Correspond à exactement un caractère

```
-- Produits dont le nom commence par "Smart"  
SELECT * FROM Produits WHERE nom_produit LIKE 'Smart%';
```

```
-- Produits dont le nom finit par "phone"  
SELECT * FROM Produits WHERE nom_produit LIKE '%phone';
```

```
-- Produits dont le nom contient "book" n'importe où  
SELECT * FROM Produits WHERE nom_produit LIKE '%book%';
```

```
-- Produits dont le nom a exactement 5 caractères  
SELECT * FROM Produits WHERE nom_produit LIKE '_____';
```

```
-- Produits dont le troisième caractère est un "b"  
SELECT * FROM Produits WHERE nom_produit LIKE '__b%';
```

4.3 Sensibilité à la casse

La sensibilité à la casse de LIKE dépend du SGBD et de la collation utilisée :

- MySQL avec collation `*_ci` (case insensitive) : LIKE n'est pas sensible à la casse
- MySQL avec collation `*_cs` (case sensitive) : LIKE est sensible à la casse
- PostgreSQL : Sensible à la casse par défaut, mais ILIKE existe pour une recherche insensible à la casse

```
-- PostgreSQL (insensible à la casse)  
SELECT * FROM Produits WHERE nom_produit ILIKE '%phone%';
```

```
-- SQL Server (insensible à la casse avec LOWER)  
SELECT * FROM Produits WHERE LOWER(nom_produit) LIKE '%phone%';
```

5. Gestion des valeurs NULL

5.1 Comprendre NULL

NULL représente une valeur inconnue ou absente, et non pas une valeur vide ou zéro. Cette distinction est cruciale pour comprendre le comportement des filtres SQL.

Caractéristiques importantes de NULL :

- NULL n'est égal à rien, pas même à lui-même (NULL = NULL donne NULL, pas TRUE)
- Toute comparaison avec NULL donne NULL (pas TRUE ou FALSE)
- NULL dans une expression logique suit la logique à trois valeurs (TRUE, FALSE, NULL)

5.2 Opérateurs IS NULL et IS NOT NULL

Pour tester si une valeur est NULL ou non, utilisez les opérateurs IS NULL et IS NOT NULL :

```
-- Trouver les produits sans catégorie
SELECT *
FROM Produits
WHERE categorie_id IS NULL;

-- Trouver les clients qui ont une ville spécifiée
SELECT *
FROM Clients
WHERE ville IS NOT NULL;
```

5.3 Pièges courants avec NULL

Erreur fréquente : utiliser l'opérateur d'égalité avec NULL

```
-- INCORRECT : ne retournera pas les lignes où categorie_id est NULL
SELECT * FROM Produits WHERE categorie_id = NULL;

-- CORRECT
SELECT * FROM Produits WHERE categorie_id IS NULL;
```

Attention aux fonctions d'agrégation qui ignorent généralement les valeurs NULL :

```
-- Comptera les lignes où prix n'est pas NULL
SELECT COUNT(prix) FROM Produits;

-- Comptera toutes les lignes, même celles où prix est NULL
SELECT COUNT(*) FROM Produits;
```

6. L'opérateur IN

6.1 Utilisation de IN pour listes de valeurs

L'opérateur IN permet de vérifier si une valeur correspond à l'une des valeurs d'une liste :

```
-- Équivalent à ville = 'Paris' OR ville = 'Lyon' OR ville = 'Marseille'
SELECT *
FROM Clients
WHERE ville IN ('Paris', 'Lyon', 'Marseille');
```

```
-- Produits dans certaines catégories
SELECT *
FROM Produits
WHERE categorie_id IN (1, 3, 5);
```

6.2 Comparaison avec OR

IN est souvent plus lisible et plus performant que plusieurs conditions OR :

```
-- Moins lisible et potentiellement moins performant
SELECT * FROM Clients
WHERE ville = 'Paris' OR ville = 'Lyon' OR ville = 'Marseille';
```

```
-- Plus concis et généralement plus efficace
SELECT * FROM Clients
WHERE ville IN ('Paris', 'Lyon', 'Marseille');
```

6.3 Utilisation de NOT IN

NOT IN vérifie qu'une valeur ne correspond à aucune des valeurs d'une liste :

```
-- Clients qui n'habitent pas dans ces villes
SELECT *
FROM Clients
WHERE ville NOT IN ('Paris', 'Lyon', 'Marseille');
```

Attention : NOT IN peut donner des résultats inattendus si la liste contient NULL.

7. L'opérateur BETWEEN

7.1 Vérifier si une valeur est dans un intervalle

L'opérateur BETWEEN vérifie si une valeur se situe dans un intervalle inclusif :

```
-- Produits dont le prix est entre 50 et 100 € (inclus)
SELECT *
FROM Produits
WHERE prix BETWEEN 50 AND 100;
```

```
-- Commandes passées au premier trimestre 2023
SELECT *
FROM Commandes
WHERE date_commande BETWEEN '2023-01-01' AND '2023-03-31';
```

7.2 Équivalence avec les opérateurs de comparaison

BETWEEN est équivalent à une combinaison de $\geq$ et $\leq$:

```
-- Ces deux requêtes sont équivalentes
SELECT * FROM Produits WHERE prix BETWEEN 50 AND 100;
SELECT * FROM Produits WHERE prix >= 50 AND prix <= 100;
```

7.3 Utilisation de NOT BETWEEN

NOT BETWEEN vérifie si une valeur est en dehors d'un intervalle :

```
-- Produits dont le prix est inférieur à 50 € ou supérieur à 100 €
SELECT *
FROM Produits
WHERE prix NOT BETWEEN 50 AND 100;
```

Prochaine étape

Dans le prochain module, nous aborderons le tri et la limitation des résultats avec ORDER BY et LIMIT, ce qui vous permettra de mieux organiser et présenter les données extraites.