Module 2 : Requêtes de base avec SELECT et FROM

ACADYS

Objectifs pédagogiques

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

- Maîtriser la syntaxe de la clause SELECT
- Sélectionner des colonnes spécifiques dans une table
- Comprendre l'utilisation correcte de la clause FROM
- Utiliser les opérateurs arithmétiques dans les requêtes
- Créer des colonnes calculées dans les résultats

1. La clause SELECT : fonction et syntaxe

1.1 Rôle de la clause SELECT

La clause SELECT est l'élément fondamental de toute requête de consultation de données en SQL. Elle permet de spécifier quelles colonnes ou expressions doivent être incluses dans les résultats de la requête.

Sa syntaxe de base est :

```
SELECT expression1, expression2, ...
FROM table;
```

Une expression peut être :

- Le nom d'une colonne
- Une constante (valeur littérale)
- Une fonction
- Une opération arithmétique
- Une expression conditionnelle

1.2 Sélection de colonnes spécifiques

L'avantage de SQL est qu'il vous permet de ne sélectionner que les colonnes dont vous avez besoin, ce qui améliore la lisibilité et les performances.

```
-- Sélectionner uniquement le nom et le prénom des clients
SELECT nom, prenom
FROM Clients;

-- Sélectionner le nom et le prix des produits
SELECT nom_produit, prix
FROM Produits;
```

1.3 Sélection de toutes les colonnes

Pour sélectionner toutes les colonnes d'une table, utilisez l'astérisque (*) :

```
-- Sélectionner toutes les colonnes de la table Clients
SELECT *
FROM Clients;
```

Important : Bien que pratique pour l'exploration, l'utilisation de `SELECT *` n'est généralement pas recommandée dans le code de production pour plusieurs raisons :

- Elle peut réduire les performances en récupérant des données inutiles
- Elle rend le code vulnérable aux modifications futures de la structure de la table
- Elle complique la lecture des résultats quand il y a beaucoup de colonnes

1.4 Ordre d'affichage des colonnes

L'ordre des colonnes dans les résultats correspond à l'ordre dans lequel vous les listez dans la clause SELECT :

```
-- Les résultats montreront d'abord le prénom puis le nom
SELECT prenom, nom
FROM Clients;

-- Les résultats montreront d'abord le nom puis le prénom
SELECT nom, prenom
FROM Clients;
```

2. La clause FROM : spécifier la source des données

2.1 Rôle de la clause FROM

La clause FROM spécifie la table (ou les tables) à partir de laquelle les données doivent être extraites. Elle est essentielle car elle indique au SGBD où trouver les colonnes mentionnées dans la clause SELECT.

```
SELECT colonne1, colonne2
FROM nom_de_la_table;
```

2.2 Sensibilité à la casse des noms de tables

La sensibilité à la casse des noms de tables dépend du système de gestion de base de données (SGBD) et de sa configuration :

- MySQL sous Windows : non sensible à la casse par défaut
- MySQL sous Linux : sensible à la casse par défaut
- PostgreSQL : sensible à la casse sauf si les noms sont entre guillemets
- SQL Server : généralement non sensible à la casse

- Oracle : généralement sensible à la casse

Bonne pratique : Pour une portabilité maximale, utilisez toujours la même casse que celle utilisée lors de la création de la table.

```
 -- Si votre table a été créée comme "Clients"
SELECT nom, prenom FROM Clients;  -- Recommandé pour la cohérence

-- Ces variantes pourraient fonctionner ou non selon le SGBD
SELECT nom, prenom FROM clients;  -- Minuscules
SELECT nom, prenom FROM CLIENTS;  -- Majuscules
```

3. Opérations arithmétiques dans SELECT

3.1 Opérateurs arithmétiques disponibles

SQL permet d'effectuer des calculs directement dans la clause SELECT :

| Opérateur | Description | Exemple | |-----------|-------------|---------| | + | Addition | SELECT prix + 10 | | - | Soustraction | SELECT prix - 5 | | * | Multiplication | SELECT prix * 1.2 | | / | Division | SELECT prix / 2 | | % | Modulo (reste de division) | SELECT quantite % 2 |

3.2 Exemples d'opérations arithmétiques

```
 -- Calculer le prix avec TVA (20%)
SELECT nom_produit, prix, prix * 1.2 AS prix_ttc
FROM Produits;

-- Calculer la valeur du stock pour chaque produit
SELECT nom_produit, prix, stock, prix * stock AS valeur_stock
FROM Produits;

-- Calculer une remise de 10%
SELECT nom_produit, prix, prix * 0.9 AS prix_remise
FROM Produits;
```

3.3 Priorité des opérations

Les opérations arithmétiques en SQL suivent les règles standard de priorité :

1. Parenthèses
2. Multiplication, division, modulo
3. Addition, soustraction

```
 -- La multiplication est effectuée avant l'addition
SELECT prix + prix * 0.2 AS prix_ttc  -- Équivalent à prix + (prix * 0.2)
FROM Produits;
```

```
-- Utilisation de parenthèses pour changer l'ordre
SELECT (prix + 10) * 0.9 AS prix_special
FROM Produits;
```

4. Colonnes calculées et expressions

4.1 Créer des colonnes calculées

Vous pouvez créer des colonnes calculées en combinant des colonnes existantes avec des opérations arithmétiques :

```
 -- Concaténer le prénom et le nom (la syntaxe peut varier selon le SGBD)
-- Pour MySQL
SELECT CONCAT(prenom, ' ', nom) AS nom_complet
FROM Clients;

-- Pour SQL Server
SELECT prenom + ' ' + nom AS nom_complet
FROM Clients;

-- Calculer la marge sur chaque produit
SELECT nom_produit, prix_vente, prix_achat,
    prix_vente - prix_achat AS marge,
    (prix_vente - prix_achat) / prix_achat * 100 AS pourcentage_marge
FROM Produits;
```

4.2 Utilisation des fonctions de chaîne

SQL offre de nombreuses fonctions pour manipuler les chaînes de caractères dans les requêtes :

```
 -- Mettre en majuscules
SELECT UPPER(nom) AS nom_majuscule
FROM Clients;

-- Extraire une partie d'une chaîne
SELECT nom_produit, SUBSTRING(nom_produit, 1, 10) AS nom_court
FROM Produits;

-- Obtenir la longueur d'une chaîne
SELECT nom_produit, LENGTH(nom_produit) AS longueur_nom
FROM Produits;
```

4.3 Utilisation de valeurs littérales

Vous pouvez inclure des valeurs littérales (constantes) dans vos requêtes SELECT :

```
 -- Ajouter une colonne constante à tous les résultats
SELECT nom_produit, prix, 'En stock' AS disponibilite
FROM Produits
WHERE stock > 0;

-- Combiner valeurs littérales et colonnes
SELECT nom_produit, CONCAT(prix, ' €') AS prix_affiche
FROM Produits;
```

5. La clause DISTINCT : éliminer les doublons

5.1 Problème des valeurs en double

Parfois, une requête peut retourner des lignes en double, particulièrement lorsque vous sélectionnez un sous-ensemble de colonnes :

```
 -- Peut retourner plusieurs fois la même ville si plusieurs clients y habitent
SELECT ville
FROM Clients;
```

5.2 Utilisation de DISTINCT

Le mot-clé DISTINCT élimine les doublons dans les résultats :

```
 -- Retourne chaque ville une seule fois
SELECT DISTINCT ville
FROM Clients;

-- Peut aussi s'appliquer à plusieurs colonnes
SELECT DISTINCT categorie_id, prix
FROM Produits;
```

Important : DISTINCT s'applique à toutes les colonnes listées dans la requête, pas seulement à la première.

6. LIMIT et TOP : limiter le nombre de résultats

6.1 Syntaxe selon les SGBD

Différents systèmes SQL utilisent différentes syntaxes pour limiter le nombre de lignes retournées :

MySQL, PostgreSQL, SQLite :

```
 -- Retourner les 5 premiers clients
SELECT *
```

```
FROM Clients
LIMIT 5;
```

SQL Server :

```
 -- Retourner les 5 premiers clients
SELECT TOP 5 *
FROM Clients;
```

Oracle (avant 12c) :

```
 -- Retourner les 5 premiers clients
SELECT *
FROM Clients
WHERE ROWNUM <= 5;
```

Oracle (12c et plus récent) :

```
 -- Retourner les 5 premiers clients
SELECT *
FROM Clients
FETCH FIRST 5 ROWS ONLY;
```

6.2 Limitation avec décalage (OFFSET)

Pour la pagination de résultats, vous pouvez combiner LIMIT avec OFFSET :

```
 -- Retourner les clients 11 à 20
SELECT *
FROM Clients
LIMIT 10 OFFSET 10;

-- Syntaxe alternative (MySQL)
SELECT *
FROM Clients
LIMIT 10, 10;
```

Lecture complémentaire

- "Mastering SQL SELECT Statements" (documentation en ligne du SGBD utilisé)
- "SQL Performance Best Practices" pour comprendre l'impact de SELECT *

Prochaine étape

Dans le prochain module, nous approfondirons l'utilisation de la clause WHERE pour filtrer les résultats selon différentes conditions.