SELECT WHERE LIKE

La commande `SELECT` est utilisée pour interroger une base de données et récupérer des données à partir d'une ou plusieurs tables. C'est l'une des commandes les plus couramment utilisées en SQL et est une partie essentielle du travail avec les bases de données.

La syntaxe de base de la commande `SELECT` est la suivante:

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

Dans cette syntaxe, `colonne1`, `colonne2`, ... sont les noms des colonnes à partir desquelles vous souhaitez récupérer des données, et `nom_de_table` est le nom de la table à partir de laquelle vous souhaitez récupérer des données. La clause `WHERE` est facultative et est utilisée pour filtrer les lignes renvoyées par la requête en fonction d'une condition spécifiée.

Voici quelques exemples d'utilisation de la commande `SELECT`:

- Pour récupérer toutes les colonnes d'une table:

```
SELECT *
FROM nom_de_table;
```

- Pour récupérer des colonnes spécifiques d'une table:

```
SELECT colonne1, colonne2, ...
FROM nom_de_table;
```

- Pour récupérer des lignes qui répondent à une condition spécifique:

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

- Pour trier les lignes renvoyées par une requête:

```
SELECT colonne1, colonne2, ...
FROM nom_de_table
ORDER BY nom_de_colonne [ASC|DESC];
```

- Pour regrouper des lignes en fonction d'une ou plusieurs colonnes et calculer des valeurs agrégées:

```
SELECT nom_de_colonne, fonction_d'agrégation(nom_de_colonne)
FROM nom_de_table
GROUP BY nom_de_colonne;
```

La commande `SELECT` prend également en charge des fonctionnalités plus avancées telles que les sous-requêtes, les jointures et les unions. Celles-ci vous permettent d'effectuer des requêtes complexes et de récupérer des données à partir de plusieurs tables dans une seule requête.

WHERE

La clause `WHERE` est utilisée dans MySQL pour filtrer les lignes renvoyées par une requête en fonction d'une ou plusieurs conditions. Elle est utilisée conjointement avec les instructions `SELECT`, `UPDATE` et `DELETE` pour spécifier quelles lignes doivent être affectées par la requête.

Voici un exemple d'utilisation de la clause `WHERE` dans une instruction `SELECT` pour récupérer des lignes qui répondent à une condition spécifique:

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

Dans cet exemple, la clause `WHERE` est utilisée pour filtrer les lignes renvoyées par l'instruction `SELECT` en fonction de la condition spécifiée. Seules les lignes qui répondent à la condition seront renvoyées par la requête.

Vous pouvez utiliser différents opérateurs de comparaison (tels que `=`, `<>`, `<`, `>`, etc.) et des opérateurs logiques (tels que `AND`, `OR` et `NOT`) pour construire des conditions complexes dans la clause `WHERE`. Voici quelques exemples:

- Pour récupérer des lignes où la valeur d'une colonne est égale à une valeur spécifique:

```
SELECT colonne1, colonne2, ...
FROM nom_de_table
WHERE nom_de_colonne = valeur;
```

- Pour récupérer des lignes où la valeur d'une colonne n'est pas égale à une valeur spécifique:

```
SELECT colonne1, colonne2, ...
FROM nom_de_table
WHERE nom_de_colonne >= valeur;
```

- Pour récupérer des lignes où la valeur d'une colonne est supérieure ou égale à une valeur spécifique:

```
SELECT colonne1, colonne2, ...
FROM nom_de_table
WHERE nom_de_colonne >= valeur;
```

- Pour récupérer des lignes où plusieurs conditions sont remplies:

```
SELECT colonne1, colonne2, ...
FROM nom_de_table
WHERE condition1 AND condition2;
```

LIKE

En MySQL, l'opérateur `LIKE` est utilisé dans la clause `WHERE` d'une instruction `SELECT`, `INSERT`, `UPDATE` ou `DELETE` pour rechercher un motif spécifié dans une colonne. Il est souvent utilisé pour rechercher des données textuelles dans une base de données.

Voici un exemple d'utilisation de l'opérateur `LIKE` pour rechercher des données textuelles dans une base de données MySQL:

```
SELECT colonne1, colonne2, ...
FROM nom_de_table
WHERE nom_de_colonne LIKE 'motif';
```

Dans cet exemple, l'opérateur `LIKE` est utilisé pour rechercher des lignes dans la table `nom_de_table` où la valeur de la colonne `nom_de_colonne` correspond au motif spécifié. Le motif peut inclure des caractères génériques tels que `%` et `_` pour représenter respectivement n'importe quel nombre de caractères ou un seul caractère.

Par exemple, pour rechercher toutes les lignes où la valeur de la colonne `nom_de_colonne` commence par la lettre 'A', vous pouvez utiliser la requête suivante:

```
SELECT colonne1, colonne2, ...
FROM nom_de_table
WHERE nom_de_colonne LIKE 'A%';
```

De même, pour rechercher toutes les lignes où la valeur de la colonne `nom_de_colonne` contient le mot 'texte', vous pouvez utiliser la requête suivante:

`SELECT colonne1, colonne2, ... FROM nom_de_table WHERE nom_de_colonne LIKE '%texte%';`

Exemple avec la base fruits_legumes

Pour extraire les fruits et légumes verts qui contiennent la lettre 'p' dans le nom, nous devons interroger à la fois les tables `fruits` et `legumes`. Voici comment vous pourriez faire cela

```
-- Sélection des fruits verts avec 'p' dans le nom
SELECT 'Fruit' AS Type, nom
FROM fruits_legumes.fruits
WHERE couleur = 'Vert' AND nom LIKE '%p%'

UNION

-- Sélection des légumes verts avec 'p' dans le nom
SELECT 'Légume' AS Type, nom
FROM fruits_legumes.legumes
WHERE couleur = 'Vert' AND nom LIKE '%p%';
```

La première partie de la requête sélectionne tous les fruits verts qui contiennent la lettre 'e' dans leur nom, tandis que la seconde partie sélectionne tous les légumes verts qui répondent aux mêmes critères. L'opérateur `UNION` combine les résultats des deux requêtes en un seul ensemble de résultats.

La colonne "Type" est ajoutée pour distinguer si chaque ligne est un fruit ou un légume.

Les Opérateurs en MySQL

MySQL offre plusieurs opérateurs qui peuvent être utilisés pour effectuer des comparaisons et des opérations logiques sur les données stockées dans les bases de données. Ces opérateurs peuvent être utilisés dans des instructions `SELECT`, `INSERT`, `UPDATE` et `DELETE` pour spécifier des conditions et filtrer les données renvoyées par une requête.

Voici quelques exemples d'opérateurs disponibles en MySQL:

- **Opérateurs de comparaison** : utilisés pour comparer des valeurs et renvoyer un résultat booléen (`TRUE` ou `FALSE`). Dans MySQL, ils incluent `=`, `<>`, `<`, `>`, `<=` et `>=`.
- **Opérateurs logiques** : utilisés pour combiner plusieurs conditions et renvoyer un résultat booléen. Dans MySQL, ils incluent `AND`, `OR` et `NOT`.
- **Opérateurs de chaîne** : utilisés pour effectuer des opérations sur des chaînes de caractères. Dans MySQL, ils incluent `CONCAT`, `LENGTH`, `LEFT`, `RIGHT`, `REPLACE` et `SUBSTRING`.

Voici un exemple d'utilisation de ces opérateurs dans une instruction `SELECT`:

```
SELECT column1, column2, ...
FROM table_name
WHERE column1 = value1 AND column2 <> value2;
```

Dans cet exemple, les opérateurs de comparaison (`=` et `<>`) sont utilisés pour spécifier des conditions dans la clause `WHERE`. L'opérateur logique `AND` est utilisé pour combiner ces conditions. Seules les lignes qui répondent à ces conditions seront renvoyées par la requête.

Les opérateurs logiques

Ils sont utilisés pour combiner plusieurs conditions dans une requête et renvoient un résultat booléen (`TRUE` ou `FALSE`) en fonction de l'évaluation des conditions combinées. Voici les opérateurs logiques disponibles en MySQL:

- `AND`: Cet opérateur renvoie `TRUE` si toutes les conditions combinées avec `AND` sont `TRUE`.

Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `nom` est égale à 'Dupont' et la valeur de la colonne `age` est supérieure à 30, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE nom = 'Dupont' AND age > 30;
```

- `OR`: Cet opérateur renvoie `TRUE` si au moins une des conditions combinées avec `OR` est `TRUE`.

Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `nom` est égale à 'Dupont' ou la valeur de la colonne `age` est supérieure à 30, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE nom = 'Dupont' OR age > 30;
```

- `NOT`: Cet opérateur inverse le résultat d'une condition.

Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `nom` n'est pas égale à 'Dupont', vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE NOT nom = 'Dupont';
```

Vous pouvez combiner ces opérateurs logiques pour construire des conditions complexes dans vos requêtes. Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `nom` est égale à 'Dupont' et la valeur de la colonne `age` est supérieure à 30 ou inférieure à 20, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE nom = 'Dupont' AND (age > 30 OR age < 20);
```

Les opérateurs de comparaison

Ils sont utilisés pour comparer des valeurs dans une requête et renvoient un résultat booléen (`TRUE` ou `FALSE`) en fonction de la comparaison effectuée. Voici les opérateurs de comparaison disponibles en MySQL:

- `=`: Cet opérateur teste l'égalité entre deux valeurs.

Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `nom` est égale à 'Dupont', vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE nom = 'Dupont';
```

- `<>` ou `!=`: Ces opérateurs testent l'inégalité entre deux valeurs.

Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `nom` n'est pas égale à 'Dupont', vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE nom <> 'Dupont';
```

- `<`: Cet opérateur teste si une valeur est inférieure à une autre valeur.

Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `age` est inférieure à 30, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE age < 30;
```

- `>`: Cet opérateur teste si une valeur est supérieure à une autre valeur. Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `age` est supérieure à 30, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE age > 30;
```

- `<=`: Cet opérateur teste si une valeur est inférieure ou égale à une autre valeur. Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `age` est inférieure ou égale à 30, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE age <= 30;
```

- `>=`: Cet opérateur teste si une valeur est supérieure ou égale à une autre valeur. Par exemple, pour sélectionner les lignes d'une table où la valeur de la colonne `age` est supérieure ou égale à 30, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM nom_de_table
WHERE age >= 30;
```

Les opérateurs de chaîne

Voici quelques exemples d'utilisation des opérateurs `CONCAT`, `LENGTH`, `LEFT`, `RIGHT`, `REPLACE` et `SUBSTRING` en MySQL:

- `CONCAT`: Cette fonction concatène plusieurs chaînes de caractères en une seule chaîne. Par exemple, pour concaténer les valeurs des colonnes `prenom` et `nom` avec un espace entre les deux, vous pouvez utiliser la requête suivante:

```
SELECT CONCAT(prenom, ' ', nom) AS nom_complet
FROM nom_de_table;
```

- `LENGTH`: Cette fonction renvoie la longueur d'une chaîne de caractères en octets. Par exemple, pour calculer la longueur de la valeur de la colonne `nom`, vous pouvez utiliser la requête suivante:

```
SELECT LENGTH(nom) AS longueur_nom
FROM nom_de_table;
```

- `LEFT`: Cette fonction renvoie les n premiers caractères d'une chaîne de caractères. Par exemple, pour renvoyer les 3 premiers caractères de la valeur de la colonne `nom`, vous pouvez utiliser la requête suivante:

```
SELECT LEFT(nom, 3) AS initiales
FROM nom_de_table;
```

- `RIGHT`: Cette fonction renvoie les n derniers caractères d'une chaîne de caractères. Par exemple, pour renvoyer les 3 derniers caractères de la valeur de la colonne `nom`, vous pouvez utiliser la requête suivante:

```
SELECT RIGHT(nom, 3) AS suffixe
FROM nom_de_table;
```

- `REPLACE`: Cette fonction remplace toutes les occurrences d'une sous-chaîne dans une chaîne de caractères par une autre sous-chaîne. Par exemple, pour remplacer toutes les occurrences de 'M.' par 'Monsieur' dans la valeur de la colonne `nom`, vous pouvez utiliser la requête suivante:

```
SELECT REPLACE(nom, 'M.', 'Monsieur') AS nom_formel
FROM nom_de_table;
```

- `SUBSTRING`: Cette fonction renvoie une sous-chaîne d'une chaîne de caractères en spécifiant la position de départ et la longueur de la sous-chaîne. Par exemple, pour renvoyer les 3 caractères à partir du 2ème caractère de la valeur de la colonne `nom`, vous pouvez utiliser la requête suivante:

```
SELECT SUBSTRING(nom, 2, 3) AS extrait
FROM nom_de_table;
```

L'opérateur BETWEEN

L'opérateur `BETWEEN` est utilisé pour récupérer des valeurs à partir d'une plage de valeurs spécifiée. Cet opérateur est utilisé avec la clause `WHERE` pour filtrer les lignes d'une table en fonction d'une condition de plage.

Voici la syntaxe de base pour utiliser l'opérateur `BETWEEN` dans une requête `SELECT`:

```
SELECT colonnes
FROM nom_de_la_table
WHERE colonne BETWEEN valeur1 AND valeur2;
```

Dans cet exemple, `colonnes` représente les colonnes que vous souhaitez récupérer, `nom_de_la_table` est le nom de la table à partir de laquelle vous souhaitez récupérer des données, `colonne` est le nom de la colonne sur laquelle vous souhaitez appliquer la condition de plage, et `valeur1` et `valeur2` sont les valeurs délimitant la plage de valeurs à récupérer.

L'opérateur `BETWEEN` renvoie les lignes pour lesquelles la valeur de la colonne spécifiée est comprise entre les deux valeurs délimitant la plage, inclusivement. Par exemple, si vous avez une table `produits` avec une colonne `prix`, et que vous souhaitez récupérer tous les produits dont le prix est compris entre 10 et 20, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM produits
WHERE prix BETWEEN 10 AND 20;
```

Cette requête renverra toutes les lignes de la table `produits` pour lesquelles la valeur de la colonne `prix` est comprise entre 10 et 20.

Vous pouvez également utiliser l'opérateur `NOT BETWEEN` pour récupérer les lignes qui ne sont pas dans une plage de valeurs spécifiée. Par exemple, pour récupérer tous les produits dont le prix n'est pas compris entre 10 et 20, vous pouvez utiliser la requête suivante:

```
SELECT *
FROM produits
WHERE prix NOT BETWEEN 10 AND 20;
```

Cette requête renverra toutes les lignes de la table `produits` pour lesquelles la valeur de la colonne `prix` n'est pas comprise entre 10 et 20.

Liste des opérateurs de MYSQL

MySQL propose une grande variété d'opérateurs pour effectuer des opérations sur les données. Voici une liste des différents types d'opérateurs disponibles dans MySQL:

1. **Opérateurs arithmétiques** : utilisés pour effectuer des opérations arithmétiques telles que l'addition, la soustraction, la multiplication, la division et le modulo.
2. **Opérateurs de comparaison** : utilisés pour comparer les valeurs entre les opérandes et retourner vrai ou faux selon la condition spécifiée dans l'instruction. Les opérateurs de comparaison incluent `>`, `<`, `=`, `!=`, `>=`, `<=`, `!<` et `!>`.
3. **Opérateurs logiques** : utilisés pour combiner plusieurs conditions dans une instruction. Les opérateurs logiques incluent `AND`, `OR`, `XOR` et `NOT`.
4. **Opérateur BETWEEN** : permet de vérifier si une valeur appartient à une plage (bornes incluses).
5. **Opérateur IN** : L'opérateur IN permet de s'assurer si une valeur est dans une liste.
6. **Opérateur LIKE** : utilisé pour effectuer des recherches de chaînes de caractères en utilisant des caractères génériques.
7. **Opérateur IS NULL** : utilisé pour vérifier si une valeur est nulle.
8. **Opérateur IS NOT NULL** : utilisé pour vérifier si une valeur n'est pas nulle.
9. **Opérateur REGEXP** : utilisé pour effectuer des recherches de chaînes de caractères en utilisant des expressions régulières.
10. **Opérateur CONCAT** : utilisé pour concaténer plusieurs chaînes de caractères en une seule chaîne.

Fonction Mathématiques dans les requêtes

MySQL offre plusieurs fonctions mathématiques qui peuvent être utilisées pour effectuer des calculs sur les données stockées dans les bases de données. Ces fonctions peuvent être utilisées dans des instructions `SELECT`, `INSERT`, `UPDATE` et `DELETE` pour effectuer des calculs sur les données renvoyées par une requête ou pour mettre à jour des données dans une table.

Voici quelques exemples de fonctions mathématiques disponibles en MySQL:

- `ABS(x)` : renvoie la valeur absolue de `x`.
- `CEILING(x)` : renvoie le plus petit entier supérieur ou égal à `x`.
- `FLOOR(x)` : renvoie le plus grand entier inférieur ou égal à `x`.
- `MOD(x, y)` : renvoie le reste de la division de `x` par `y`.
- `POWER(x, y)` : renvoie la puissance de `x` élevée à la puissance `y`.
- `RAND()` : renvoie un nombre aléatoire compris entre 0 et 1.
- `ROUND(x, d)` : renvoie le nombre `x` arrondi à `d` décimales.
- `SQRT(x)` : renvoie la racine carrée de `x`.

Pour avoir la liste complète des fonctions mathématiques:

https://sql.sh/fonctions/mathematiques.

Voici un exemple d'utilisation de ces fonctions mathématiques dans une instruction `SELECT`:

```
SELECT column1, ABS(column2), CEILING(column3), FLOOR(column4)
FROM table_name;
```

Dans cet exemple, les fonctions mathématiques sont utilisées pour calculer la valeur absolue, le plafond et le plancher des valeurs des colonnes 2, 3 et 4, respectivement. Les résultats de ces calculs sont renvoyés dans les colonnes correspondantes du résultat de la requête.

Les Alias

C'est un nom temporaire donné à une colonne ou à une table dans une requête. Ils sont souvent utilisés pour améliorer la lisibilité d'une requête en donnant des noms plus descriptifs aux colonnes ou aux tables.

Voici comment vous pouvez utiliser des alias pour les colonnes dans une instruction `SELECT`:

```
SELECT column_name AS alias_name
FROM table_name;
```

Dans cet exemple, l'alias `alias_name` est donné à la colonne `column_name`. Lorsque les résultats de la requête sont renvoyés, le nom de la colonne sera remplacé par l'alias dans les résultats.

Voici un exemple concret d'utilisation d'un alias pour une colonne:

```
SELECT first_name AS prenom, last_name AS nom
FROM employees;
```

Dans cet exemple, les alias `prenom` et `nom` sont donnés aux colonnes `first_name` et `last_name`, respectivement. Lorsque les résultats de la requête sont renvoyés, les noms des colonnes seront remplacés par les alias dans les résultats.

Vous pouvez également utiliser des alias pour les tables dans une requête. Voici comment vous pouvez utiliser un alias pour une table dans une instruction `SELECT`:

```
SELECT column_name
FROM table_name AS alias_name;
```

Dans cet exemple, l'alias `alias_name` est donné à la table `table_name`. Vous pouvez ensuite utiliser cet alias pour faire référence à la table dans le reste de la requête.

Voici un exemple concret d'utilisation d'un alias pour une table:

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees AS e
JOIN departments AS d
ON e.department_id = d.department_id;
```

Dans cet exemple, les alias `e` et `d` sont donnés aux tables `employees` et `departments`, respectivement. Ces alias sont ensuite utilisés pour faire référence aux tables dans le reste de la requête.

Voici un exemple de requête qui utilise un alias pour une somme calculée en MySQL:

```
SELECT SUM(nom_de_colonne) AS total
FROM nom_de_table;
```

Dans cet exemple, la fonction `SUM` est utilisée pour calculer la somme des valeurs de la colonne `nom_de_colonne` de la table `nom_de_table`. Le mot-clé `AS` est utilisé pour donner un alias à la somme calculée, qui est `total` dans ce cas.

Le résultat de cette requête pourrait ressembler à ceci:

```
+-------+
| total |
+-------+
|   100 |
+-------+
```

Dans ce schéma de sortie, la colonne `total` contient la somme calculée des valeurs de la colonne `nom_de_colonne`. L'alias `total` est utilisé comme nom de colonne dans les résultats de la requête.

Résumé

Les alias vous permettent :

- de raccourcir vos requêtes
- de changer les noms des requêtes calculées pour améliorer la visibilité des résultats.

Les relations entre les tables de données

Elles sont utilisées pour lier les données stockées dans différentes tables et sont définies en utilisant des clés étrangères, qui sont des colonnes dans une table qui font référence à la clé primaire d'une autre table. Il existe trois types de relations entre les tables de données:

- la relation un-à-un
- la relation un-à-plusieurs
- la relation plusieurs-à-plusieurs.

➧ **Relation un-à-un** :

C'est une relation dans laquelle chaque enregistrement d'une table est associé à un seul enregistrement d'une autre table. Par exemple, supposons que vous ayez une table `employees` et une table `employeeDetails`. Chaque employé a des détails uniques stockés dans la table `employeeDetails`. Dans ce cas, vous pouvez utiliser une relation un-à-un pour lier les deux tables.

➧ **Relation un-à-plusieurs** :

C'est une relation dans laquelle chaque enregistrement d'une table peut être associé à plusieurs enregistrements d'une autre table. Par exemple, supposons que vous ayez une table `customers` et une table `orders`. Un client peut avoir plusieurs commandes, mais chaque commande ne peut être associée qu'à un seul client. Dans ce cas, vous pouvez utiliser une relation un-à-plusieurs pour lier les deux tables.

➧ **Relation plusieurs-à-plusieurs** :

C'est une relation dans laquelle plusieurs enregistrements d'une table peuvent être associés à plusieurs enregistrements d'une autre table. Par exemple, supposons que vous ayez une table `students` et une table `courses`. Un étudiant peut s'inscrire à plusieurs cours et chaque cours peut avoir plusieurs étudiants inscrits. Dans ce cas, vous pouvez utiliser une relation plusieurs-à-plusieurs pour lier les deux tables.

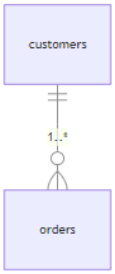
Pour créer des relations entre les tables de données, vous utilisez l'instruction `FOREIGN KEY` lors de la création des tables. Par exemple, pour créer une relation un-à-plusieurs entre les tables `customers` et `orders`, vous pouvez utiliser l'instruction suivante:

```
CREATE TABLE customers (
    customerID INT PRIMARY KEY,
    customerName VARCHAR(255)
);

CREATE TABLE orders (
    orderID INT PRIMARY KEY,
    orderDate DATE,
    customerID INT,
    FOREIGN KEY (customerID) REFERENCES customers(customerID)
);
```

Dans cet exemple, nous avons créé deux tables: `customers` et `orders`. La colonne `customerID` dans la table `orders` est définie comme clé étrangère qui fait référence à la colonne `customerID` dans la table `customers`. Cela crée une relation un-à-plusieurs entre les deux tables.

```
+------------+     +------------+
| customers  |     |   orders   |
+------------+     +------------+
| customerID |-----| customerID |
| customerName|    | orderID    |
+------------+     | orderDate  |
                   +------------+
```



Dans cet exemple, nous avons deux tables: `customers` et `orders`. La table `customers` a deux colonnes: `customerID` et `customerName`. La table `orders` a trois colonnes: `orderID`, `orderDate` et `customerID`.

La colonne `customerID` dans la table `orders` est une clé étrangère qui fait référence à la colonne `customerID` dans la table `customers`. Cela crée une relation un-à-plusieurs entre les deux tables, où chaque client peut avoir plusieurs commandes, mais chaque commande ne peut être associée qu'à un seul client.

Exemples pour chaque type de relation entre les tables de données:

Relation un-à-un :

Supposons que vous ayez une entreprise avec des employés et que vous souhaitiez stocker des informations sur les employés et leurs détails personnels dans une base de données. Vous pouvez créer deux tables: une table `employees` pour stocker les informations de base sur les employés, et une table `employeeDetails` pour stocker les détails personnels des employés. Chaque employé a un seul enregistrement dans la table `employeeDetails`, donc vous pouvez utiliser une relation un-à-un pour lier les deux tables. Voici comment vous pouvez créer ces tables en utilisant SQL:

```
CREATE TABLE employees (
    employeeID INT PRIMARY KEY,
    firstName VARCHAR(255),
    lastName VARCHAR(255)
);

CREATE TABLE employeeDetails (
    employeeID INT PRIMARY KEY,
    address VARCHAR(255),
    phoneNumber VARCHAR(255),
    FOREIGN KEY (employeeID) REFERENCES employees(employeeID)
);
```

Dans cet exemple, nous avons créé deux tables: `employees` et `employeeDetails`. La colonne `employeeID` dans la table `employeeDetails` est définie comme clé étrangère qui fait référence à la colonne `employeeID` dans la table `employees`. Cela crée une relation un-à-un entre les deux tables.

Relation un-à-plusieurs :

Supposons que vous ayez une entreprise de vente en ligne et que vous souhaitiez stocker des informations sur les clients et leurs commandes dans une base de données. Vous pouvez créer deux tables: une table `customers` pour stocker les informations sur les clients, et une table `orders` pour stocker les informations sur les commandes. Un client peut avoir plusieurs commandes, mais chaque commande ne peut être associée qu'à un seul client, donc vous pouvez utiliser une relation un-à-plusieurs pour lier les deux tables. Voici comment vous pouvez créer ces tables en utilisant SQL:

```
CREATE TABLE customers (
    customerID INT PRIMARY KEY,
    customerName VARCHAR(255)
);

CREATE TABLE orders (
    orderID INT PRIMARY KEY,
    orderDate DATE,
    customerID INT,
    FOREIGN KEY (customerID) REFERENCES customers(customerID)
);
```

Dans cet exemple, nous avons créé deux tables: `customers` et `orders`. La colonne `customerID` dans la table `orders` est définie comme clé étrangère qui fait référence à la colonne `customerID` dans la table `customers`. Cela crée une relation un-à-plusieurs entre les deux tables.

Relation plusieurs-à-plusieurs :

Supposons que vous ayez une école et que vous souhaitiez stocker des informations sur les étudiants et les cours dans une base de données. Vous pouvez créer deux tables: une table `students` pour stocker les informations sur les étudiants, et une table `courses` pour stocker les informations sur les cours. Un étudiant peut s'inscrire à plusieurs cours et chaque cours peut avoir plusieurs étudiants inscrits, donc vous pouvez utiliser une relation plusieurs-à-plusieurs pour lier les deux tables. Cependant, contrairement aux relations un-à-un et un-à-plusieurs, vous ne pouvez pas créer directement une relation plusieurs-à-plusieurs entre deux tables en utilisant des clés étrangères. Au lieu de cela, vous devez créer une troisième table appelée table d'association pour lier les deux tables. Voici comment vous pouvez créer ces tables en utilisant SQL:

```
CREATE TABLE students (
    studentID INT PRIMARY KEY,
    studentName VARCHAR(255)
);

CREATE TABLE courses (
    courseID INT PRIMARY KEY,
    courseName VARCHAR(255)
);

CREATE TABLE studentCourses (
    studentID INT,
    courseID INT,
    PRIMARY KEY (studentID, courseID),
    FOREIGN KEY (studentID) REFERENCES students(studentID),
    FOREIGN KEY (courseID) REFERENCES courses(courseID)
);
```

Dans cet exemple, nous avons créé trois tables: `students`, `courses` et `studentCourses`. La table `studentCourses` est la table d'association qui lie les deux autres tables. Elle a deux colonnes: `studentID` et `courseID`, qui sont toutes deux définies comme clés étrangères faisant référence aux colonnes `studentID` dans la table `students` et `courseID` dans la table `courses`, respectivement. Cela crée une relation plusieurs-à-plusieurs entre les deux tables.

Créer des vues

Les vues sont des requêtes nommées, stockées dans le catalogue de la base de données. Pour créer une nouvelle vue, vous utilisez l'instruction `CREATE VIEW`. Cette instruction crée une vue basée sur la requête spécifiée. Par exemple, pour créer une vue appelée `customerPayments` basée sur une requête qui renvoie des données à partir de deux tables `customers` et `payments` en utilisant une jointure interne, vous pouvez utiliser l'instruction suivante:

```
CREATE VIEW customerPayments AS
SELECT customerName, checkNumber, paymentDate, amount
FROM customers
INNER JOIN payments USING (customerNumber);
```

Une fois que vous avez exécuté l'instruction `CREATE VIEW`, MySQL crée la vue et la stocke dans la base de données. Vous pouvez maintenant référencer la vue comme une table dans les instructions SQL. Par exemple, vous pouvez interroger les données à partir de la vue `customerPayments` en utilisant l'instruction `SELECT`:

```
SELECT * FROM customerPayments;
```

La syntaxe est beaucoup plus simple. Notez qu'une vue ne stocke pas physiquement les données. Lorsque vous émettez l'instruction `SELECT` contre la vue, MySQL exécute la requête sous-jacente spécifiée dans la définition de la vue et renvoie l'ensemble des résultats. Pour cette raison, parfois, une vue est appelée une table virtuelle.

MySQL vous permet de créer une vue basée sur une instruction `SELECT` qui récupère des données à partir d'une ou plusieurs tables. Cette image illustre une vue basée sur les colonnes de plusieurs tables:

En outre, MySQL permet même de créer une vue qui ne fait pas référence à une table. Mais vous trouverez rarement ce type de vue en pratique. Par exemple, vous pouvez créer une vue appelée `daysofweek` qui renvoie les 7 jours d'une semaine en exécutant la requête suivante:

```
CREATE VIEW daysofweek (day) AS
SELECT 'Mon'
UNION SELECT 'Tue'
UNION SELECT 'Web'
UNION SELECT 'Thu'
UNION SELECT 'Fri'
UNION SELECT 'Sat'
UNION SELECT 'Sun';
```

Et vous pouvez interroger les données à partir de la vue `daysofweek` comme suit:

```
SELECT * FROM daysofweek;
```

Les avantages des vues MySQL sont les suivants:

1. **Simplifier les requêtes complexes** : Si vous avez une requête complexe fréquemment utilisée, vous pouvez créer une vue basée dessus afin de faire référence à la vue en utilisant une simple instruction `SELECT` au lieu de taper la requête à nouveau.
2. **Masquer certaines données à certains utilisateurs** : Par exemple, vous pouvez créer une vue qui ne renvoie que les colonnes d'une table auxquelles un utilisateur spécifique a accès. En SQL, les protections d'une vue ne sont pas forcément les mêmes que celles des tables sous-jacentes.
3. **Réduire les erreurs de codage** : en éliminant la nécessité de taper des requêtes complexes à plusieurs reprises.

Exemple de requête complexe simplifiée avec une vue

Supposons que vous ayez une base de données de gestion des commandes avec les tables `orders`, `orderDetails`, `products` et `customers`. Vous souhaitez créer un rapport qui affiche les informations suivantes pour chaque commande: numéro de commande, date de commande, nom du client, nombre total d'articles commandés et montant total de la commande.

Sans utiliser de vue, vous devriez écrire une requête complexe qui joint les quatre tables et utilise des fonctions d'agrégation pour calculer le nombre total d'articles et le montant total de la commande. La requête ressemblerait à ceci:

```
SELECT o.orderNumber, o.orderDate, c.customerName,
       SUM(od.quantityOrdered) AS totalQuantity,
       SUM(od.quantityOrdered * od.priceEach) AS totalPrice
FROM orders o
INNER JOIN customers c ON o.customerNumber = c.customerNumber
INNER JOIN orderDetails od ON o.orderNumber = od.orderNumber
INNER JOIN products p ON od.productCode = p.productCode
GROUP BY o.orderNumber;
```

Comme vous pouvez le voir, cette requête est assez complexe et peut être difficile à comprendre pour quelqu'un qui n'est pas familier avec la structure de la base de données.

Cependant, vous pouvez simplifier cette requête en créant une vue appelée `orderSummary` basée sur cette requête:

```
CREATE VIEW orderSummary AS
SELECT o.orderNumber, o.orderDate, c.customerName,
       SUM(od.quantityOrdered) AS totalQuantity,
       SUM(od.quantityOrdered * od.priceEach) AS totalPrice
FROM orders o
INNER JOIN customers c ON o.customerNumber = c.customerNumber
INNER JOIN orderDetails od ON o.orderNumber = od.orderNumber
INNER JOIN products p ON od.productCode = p.productCode
GROUP BY o.orderNumber;
```

Une fois que vous avez créé la vue `orderSummary`, vous pouvez interroger les données à partir de cette vue en utilisant une simple instruction `SELECT`:

```
SELECT * FROM orderSummary;
```

Comme vous pouvez le voir, l'utilisation d'une vue simplifie grandement la requête et la rend beaucoup plus facile à comprendre.

Plus d'infos sur les vues

https://apprendre-php.com/tutoriels/tutoriel-28-mysql-5-0-les-vues.html