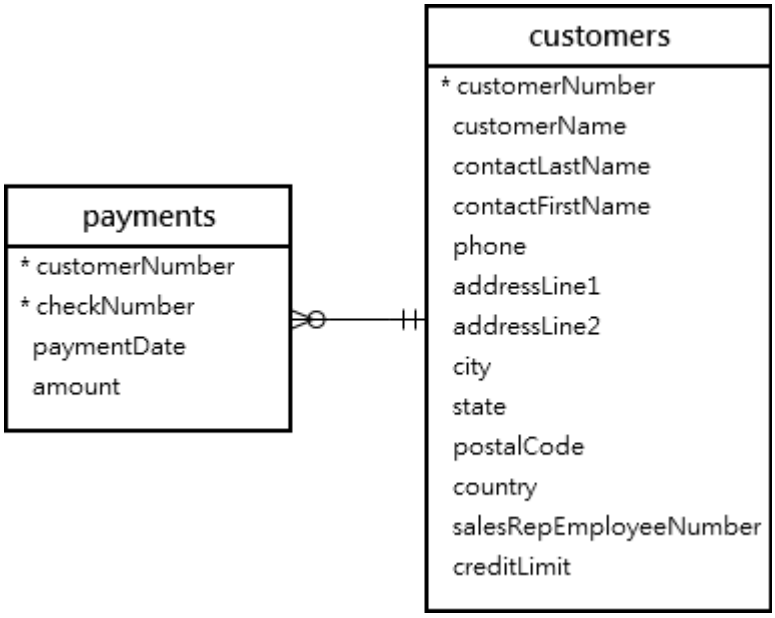
Les vues MySQL

https://dev.mysql.com/doc/refman/8.4/en/views.html

Réalisation Stéphane Pontonnier @2024

Introduction

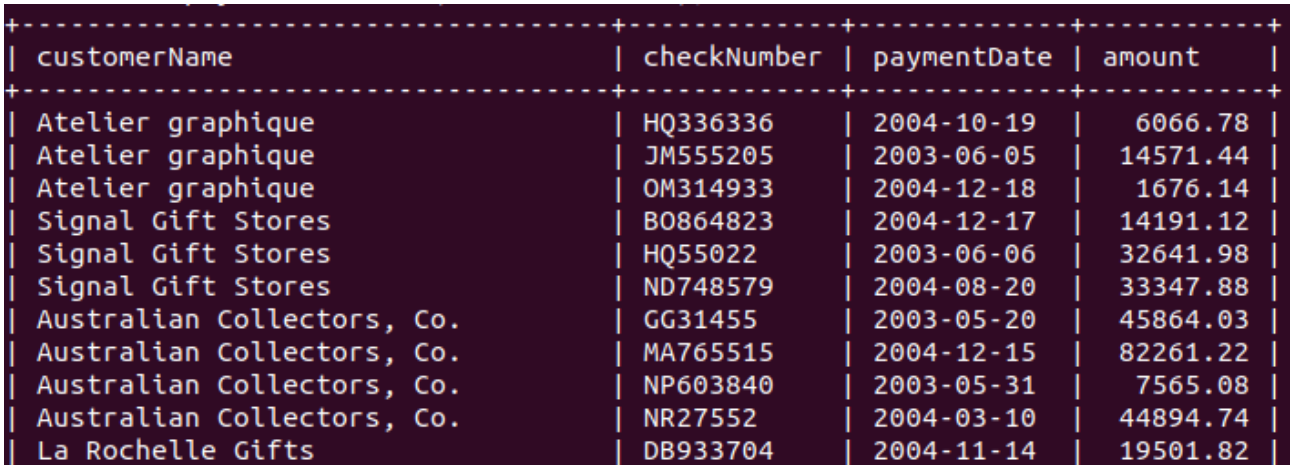
Prenons les tables suivantes : ***customers*** et ***paiements*** de la base de données.



Cette requête renvoie les données des deux tables en utilisant une jointure interne :

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

Le résultat en sortie :



customerName	checkNumber	paymentDate	amount
Atelier graphique	HQ336336	2004-10-19	6066.78
Atelier graphique	JM555205	2003-06-05	14571.44
Atelier graphique	OM314933	2004-12-18	1676.14
Signal Gift Stores	BO864823	2004-12-17	14191.12
Signal Gift Stores	HQ55022	2003-06-06	32641.98
Signal Gift Stores	ND748579	2004-08-20	33347.88
Australian Collectors, Co.	GG31455	2003-05-20	45864.03
Australian Collectors, Co.	MA765515	2004-12-15	82261.22
Australian Collectors, Co.	NP603840	2003-05-31	7565.08
Australian Collectors, Co.	NR27552	2004-03-10	44894.74
La Rochelle Gifts	DB933704	2004-11-14	19501.82

La prochaine fois, si vous souhaitez obtenir les mêmes informations, notamment le nom du client, le numéro de chèque, la date de paiement et le montant, vous devrez relancer la même requête.

Pour ce faire, vous pouvez enregistrer la requête dans un fichier, soit .txt, soit .sql, afin de pouvoir l'ouvrir et l'exécuter ultérieurement à partir de MySQL Workbench ou de tout autre outil client MySQL.

Une meilleure façon de procéder consiste à enregistrer la requête dans le serveur de base de données et à lui attribuer un nom. Cette requête nommée est appelée « vue de la base de données », ou plus simplement, « vue ».

Par définition, une vue est une requête nommée stockée dans le catalogue de la base de données.

Pour créer une nouvelle vue, vous utilisez l'instruction **CREATE VIEW**. Cette instruction crée une vue ***clientsPaiements*** basée sur la requête ci-dessus :

```
CREATE VIEW clientPaiements
AS
SELECT
    customerName,
    checkNumber,
    paymentDate,
```

```
    amount
FROM
    customers
INNER JOIN
    payments USING (customerNumber);
```

Après avoir exécuté l'instruction **CREATE VIEW**, MySQL crée la vue et la stocke dans la base de données.

Vous pouvez maintenant référencer la vue en tant que table dans les instructions SQL.

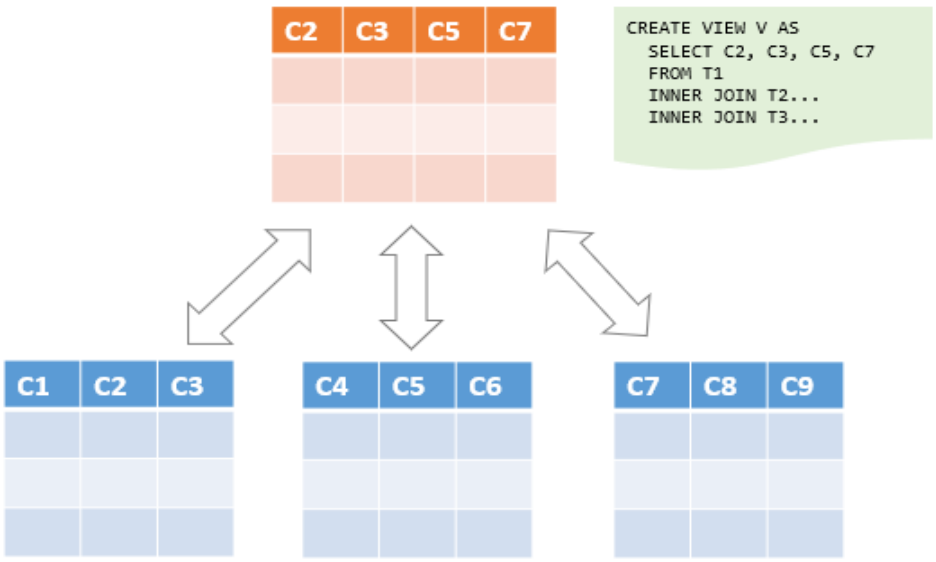
Par exemple, vous pouvez interroger les données de la vue ***clientPaiements*** à l'aide de l'instruction SELECT :

```
SELECT * FROM clientPaiements ;
```

Comme vous pouvez le constater, la syntaxe est beaucoup plus simple.

Notez qu'une vue ne stocke pas physiquement les données. Lorsque vous lancez l'instruction SELECT sur la vue, MySQL exécute la requête sous-jacente spécifiée dans la définition de la vue et renvoie l'ensemble des résultats. C'est pour cette raison que l'on parle parfois d'une vue comme d'une table virtuelle.

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

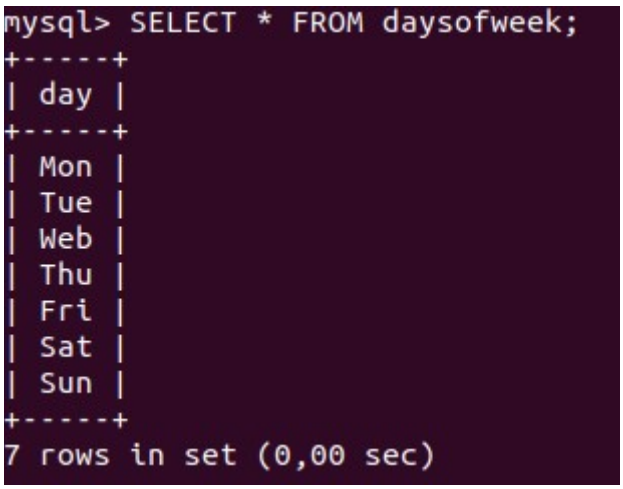


En outre, MySQL vous permet même de créer une vue qui ne fait référence à aucune table. Mais vous trouverez rarement ce type de vue dans la pratique.

Par exemple, vous pouvez créer une vue appelée ***daysofweek*** qui renvoie les 7 jours de la 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';
```

Le résultat en sortie :



Avantages des vues MySQL

Les vues MySQL offrent les avantages suivants.

1) Simplifier les requêtes complexes

Les vues permettent de simplifier les requêtes complexes. Si vous avez une requête complexe fréquemment utilisée, vous pouvez créer une vue basée sur celle-ci afin de pouvoir y faire référence à l'aide d'une simple instruction SELECT au lieu de saisir la requête une nouvelle fois.

2) Rendre la logique métier cohérente

Supposons que vous deviez répéter la même formule dans chaque requête. Ou bien vous avez une requête dont la logique métier est complexe. Pour que cette logique soit cohérente d'une requête à l'autre, vous pouvez utiliser une vue pour stocker le calcul et masquer la complexité.

3) Ajouter des couches de sécurité supplémentaires

Une table peut exposer un grand nombre de données, y compris des données sensibles telles que des informations personnelles et bancaires.

En utilisant des vues et des privilèges, vous pouvez limiter les données auxquelles les utilisateurs peuvent accéder en ne leur exposant que les données nécessaires.

Par exemple, la table ***employees*** peut contenir des informations sur le numéro de sécurité sociale et l'adresse, qui ne doivent être accessibles qu'au service des ressources humaines.

Pour exposer des informations générales telles que le prénom, le nom de famille et le sexe au service Administration générale (AG), vous pouvez créer une vue basée sur ces colonnes et accorder cette vue aux utilisateurs du service AG, et non à l'ensemble de la table ***employees***.

4) Permettre la rétrocompatibilité

Dans les systèmes existants, les vues peuvent permettre une compatibilité ascendante.

Supposons que vous souhaitiez normaliser une grande table en plusieurs tables plus petites. Et vous ne voulez pas impacter les applications actuelles qui font référence à la table.

Dans ce cas, vous pouvez créer une vue dont le nom est identique à celui de la table basée sur les nouvelles tables afin que toutes les applications puissent référencer la vue comme s'il s'agissait d'une table.

Notez qu'une vue et une table ne peuvent pas avoir le même nom. Vous devez donc supprimer la table avant de créer une vue dont le nom est identique à celui de la table supprimée.

MySQL CREATE VIEW

Introduction

L'instruction **CREATE VIEW** crée une nouvelle vue dans la base de données. La syntaxe de base :

```
CREATE [OR REPLACE] VIEW [db_name.]view_name [(column_list)]
AS
  instruction select;
```

Explications :

Tout d'abord, indiquez le nom de la vue que vous souhaitez créer après les mots-clés **CREATE VIEW**. Le nom de la vue est unique dans une base de données.

Comme les vues et les tables d'une même base de données partagent le même espace de noms, le nom d'une vue ne peut pas être le même que celui d'une table existante.

Deuxièmement, utilisez l'option **OR REPLACE** si vous souhaitez remplacer une vue existante. Si la vue n'existe pas, l'option **OR REPLACE** n'a aucun effet.

Troisièmement, spécifiez une liste de colonnes pour la vue. Par défaut, les colonnes de la vue sont dérivées de la liste de sélection de l'instruction **SELECT**. Toutefois, vous pouvez spécifier explicitement la liste des colonnes de la vue en les énumérant entre parenthèses après le nom de la vue.

Enfin, spécifiez une instruction **SELECT** qui définit la vue. L'instruction **SELECT** peut interroger des données provenant de

tables ou de vues. MySQL vous permet d'utiliser la clause **ORDER BY** dans l'instruction **SELECT**, mais l'ignore si vous effectuez une sélection dans la vue à l'aide d'une requête qui possède sa propre clause **ORDER BY**.

Par défaut, l'instruction **CREATE VIEW** crée une vue dans la base de données actuelle. Si vous souhaitez créer explicitement une vue dans une base de données donnée, vous pouvez qualifier le nom de la vue avec le nom de la base de données.

Exercices :

1/ Création d'une vue simple

A partir de la table ***orderDetails***

Créez une vue, ***salePerOrder***, qui représente les ventes totales par commande.

Si vous utilisez la commande **SHOW TABLES** pour afficher toutes les tables de la base de données ***classicmodels***, vous verrez que ***salesPerOrder*** apparaît dans la liste.

```
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_classicmodels |
+-------------------------+
| VentesChanges           |
| WorkCenterStats         |
| WorkCenters             |
| billings                |
| clientPaiements         |
| customers               |
| daysofweek              |
| employees               |
| item_changes            |
| items                   |
| members                 |
| offices                 |
| orderdetails            |
| orders                  |
| payments                |
| productlines            |
| products                |
| reminders               |
| salaireBudget           |
| salaires                |
| salePerOrder            |
| ventes                  |
+-------------------------+
22 rows in set (0,01 sec)
```

En effet, les vues et les tables partagent le même espace de noms, comme indiqué précédemment.

Pour savoir quel objet est une vue ou une table, vous utilisez la commande **SHOW FULL TABLES** comme suit :

```
mysql> SHOW FULL TABLES;
+--------------------------+------------+
| Tables_in_classicmodels | Table_type |
+--------------------------+------------+
| VentesChanges            | BASE TABLE |
| WorkCenterStats          | BASE TABLE |
| WorkCenters              | BASE TABLE |
| billings                 | BASE TABLE |
| clientPaiements          | VIEW       |
| customers                | BASE TABLE |
| daysofweek               | VIEW       |
| employees                | BASE TABLE |
| item_changes             | BASE TABLE |
| items                    | BASE TABLE |
| members                  | BASE TABLE |
| offices                  | BASE TABLE |
| orderdetails             | BASE TABLE |
| orders                   | BASE TABLE |
| payments                 | BASE TABLE |
| productlines             | BASE TABLE |
| products                 | BASE TABLE |
| reminders                | BASE TABLE |
| salaireBudget            | BASE TABLE |
| salaires                 | BASE TABLE |
| salePerOrder             | VIEW       |
| ventes                   | BASE TABLE |
+--------------------------+------------+
22 rows in set (0,01 sec)
```

La colonne Table_type de l'ensemble de résultats spécifie le type de l'objet : vue ou table (table de base).

Si vous souhaitez interroger le total des ventes pour chaque commande, il vous suffit d'exécuter une simple instruction **SELECT** sur la vue ***SalePerOrder*** comme suit :

Le résultat (partiel) en sortie :

```
mysql> SELECT * FROM salePerOrder;
+-------------+----------+
| orderNumber | total    |
+-------------+----------+
|       10165 | 67392.85 |
|       10287 | 61402.00 |
|       10310 | 61234.67 |
|       10212 | 59830.55 |
```

2/ Création d'une vue basée sur une autre vue

MySQL vous permet de créer une vue basée sur une autre vue.

Par exemple, créez une vue appelée ***bigSalesOrder*** basée sur la vue ***salesPerOrder*** pour afficher toutes les commandes clients dont le total est supérieur à 60 000 :

Le résultat en sortie :

```
mysql> SELECT
    ->     orderNumber,
    ->     total
    -> FROM
    ->     bigSalesOrder;
+-------------+----------+
| orderNumber | total    |
+-------------+----------+
|       10165 | 67392.85 |
|       10287 | 61402.00 |
|       10310 | 61234.67 |
+-------------+----------+
3 rows in set (0,00 sec)
```

3/ Création d'une vue avec jointure

Créez une vue ***customersOrders***

La somme totale des commandes de chaque client

Le résultat en sortie :

```
mysql> SELECT * FROM customerOrders
    -> ORDER BY total DESC;
+-------------+-----------------------------------+----------+
| orderNumber | customerName                      | total    |
+-------------+-----------------------------------+----------+
|       10165 | Dragon Souveniers, Ltd.           | 67392.85 |
|       10287 | Vida Sport, Ltd                   | 61402.00 |
|       10310 | Toms Spezialitäten, Ltd           | 61234.67 |
|       10212 | Euro+ Shopping Channel            | 59830.55 |
|       10207 | Diecast Collectables              | 59265.14 |
|       10127 | Muscle Machine Inc                | 58841.35 |
|       10204 | Muscle Machine Inc                | 58793.53 |
|       10126 | Corrida Auto Replicas, Ltd        | 57131.92 |
|       10222 | Collectable Mini Designs Co.      | 56822.65 |
```

4/ Créer une vue avec une sous-requête

Créer une vue ***aboveAvgProducts*** dont l'instruction **SELECT** utilise une sous-requête (une autre requête **SELECT** dans la clause **WHERE**). La vue contient les produits dont les prix d'achat sont supérieurs au prix moyen de tous les produits.

Le résultat en sortie :

```
mysql> SELECT * FROM aboveAvgProducts;
+-------------+-----------------------------------------+----------+
| productCode | productName                             | buyPrice |
+-------------+-----------------------------------------+----------+
| S10_4962    | 1962 LanciaA Delta 16V                  |   103.42 |
| S18_2238    | 1998 Chrysler Plymouth Prowler          |   101.51 |
| S10_1949    | 1952 Alpine Renault 1300                |    98.58 |
| S24_3856    | 1956 Porsche 356A Coupe                 |    98.30 |
| S12_1108    | 2001 Ferrari Enzo                       |    95.59 |
| S12_1099    | 1968 Ford Mustang                       |    95.34 |
| S18_1984    | 1995 Honda Civic                        |    93.89 |
```

5/ Création d'une vue avec des colonnes explicites

Créez une nouvelle vue basée sur les tables ***customers*** et ***orders*** avec des colonnes de vue explicites.

La vue permet d’afficher le nombre de commande total par client

Le résultat en sortie :

```
mysql> SELECT
    ->     customerName,
    ->     orderCount
    -> FROM
    ->     customerOrderStats
    -> ORDER BY
    -> orderCount,
    ->     customerName;
+-----------------------------------+------------+
| customerName                      | orderCount |
+-----------------------------------+------------+
| Bavarian Collectables Imports, Co. |          1 |
| Amica Models & Co.                |          2 |
| Auto Associés & Cie.              |          2 |
| Boards & Toys Co.                 |          2 |
| CAF Imports                       |          2 |
| Cambridge Collectables Co.        |          2 |
| Canadian Gift Exchange Network    |          2 |
| Classic Gift Ideas, Inc           |          2 |
```

Créer des vues actualisables dans MySQL

Regardons comment créer une vue actualisable et mettre à jour les données de la table sous-jacente à travers la vue.

Introduction

Dans MySQL, les vues sont non seulement interrogeables, mais aussi actualisables. Cela signifie que vous pouvez utiliser l'instruction **INSERT** ou **UPDATE** pour ajouter ou modifier des lignes de la table de base par le biais de la vue actualisable.

En outre, vous pouvez utiliser l'instruction **DELETE** pour supprimer des lignes de la table sous-jacente via la vue.

Toutefois, pour créer une vue actualisable, l'instruction **SELECT** définissant la vue ne doit contenir aucun des éléments suivants :

- Fonctions d'agrégation telles que **MIN**, **MAX**, **SUM**, **AVG** et **COUNT**.
- **DISTINCT**
- Clause **GROUP BY**.
- La clause **HAVING**.
- La clause **UNION** ou **UNION ALL**.
- Jointure gauche ou jointure externe.
- Sous-requête dans la clause SELECT ou dans la clause WHERE qui fait référence à la table apparue dans la clause FROM.
- Référence à des vues non actualisables dans la clause FROM.
- Utilisation de valeurs littérales.
- Références multiples à n'importe quelle colonne de la table de base.

Exemple de vue actualisable MySQL

Exercices :

créez une vue nommée ***officeInfo*** basée sur la table ***offices*** de la base de données. La vue fait référence à trois colonnes de la table offices : *officeCode*, *phone* et *city*.

Le résultat en sortie :

```
mysql> SELECT * FROM officeInfo;
+------------+------------------+---------------+
| officeCode | phone            | city          |
+------------+------------------+---------------+
| 1          | +1 650 219 4782  | San Francisco |
| 2          | +1 215 837 0825  | Boston        |
| 3          | +1 212 555 3000  | NYC           |
| 4          | +33 14 723 4404  | Paris         |
| 5          | +81 33 224 5000  | Tokyo         |
| 6          | +61 2 9264 2451  | Sydney        |
| 7          | +44 20 7877 2041 | London        |
+------------+------------------+---------------+
7 rows in set (0,00 sec)
```

Modifiez le numéro de téléphone du bureau avec le *officeCode* 4

dans la vue ***officeInfo*** à l'aide de l'instruction **UPDATE**.

Affichez les données de la vue ***officeInfo*** pour vérifier le changement.

Vérification des informations sur les vues actualisables

Vous pouvez vérifier si une vue d'une base de données peut être mise à jour en interrogeant la colonne *is_updatable* de la table ***views*** de la base de données ***information_schema :***

```
mysql> SELECT
    ->     table_name,
    ->     is_updatable
    -> FROM
    ->     information_schema.views
    -> WHERE
    ->     table_schema = 'classicmodels';
+--------------------+--------------+
| TABLE_NAME         | IS_UPDATABLE |
+--------------------+--------------+
| aboveAvgProducts   | NO           |
| bigSalesOrder      | NO           |
| clientPaiements    | YES          |
| customerOrderStats | NO           |
| customerOrders     | NO           |
| daysofweek         | NO           |
| officeInfo         | YES          |
| salePerOrder       | NO           |
+--------------------+--------------+
8 rows in set (0,01 sec)
```

Suppression de lignes dans la vue

Tout d'abord, créez une table nommée ***item*** (id INT AUTO-INCREMENT, name VARCHAR(100) NOT NULL, price DECIMAL(11,2) NOT NULL, PRIMARY KEY(id) ;)

insérez quelques lignes dans la table ***items*** et créez une vue qui contient les items dont les prix sont supérieurs à 700.

Interrogez la vue pour voir les données qu'elle contient.

Utilisez l'instruction **DELETE** pour supprimer la ligne dont l'id est 3 dans la vue.

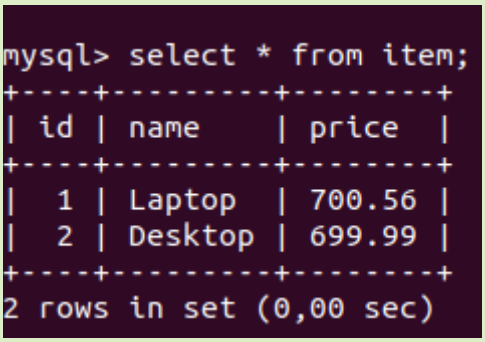
MySQL renvoie un message indiquant que 1 ligne(s) est affectée.

Vérifiez à nouveau les données dans la vue

Enfin, interrogez les données des éléments de la table ***item*** pour

vérifier si l'instruction **DELETE** a supprimé la ligne.

La sortie montre que la ligne avec l'identifiant 3 a été supprimée de la table de base.



Renommer une vue

Introduction

Dans MySQL, les vues et les tables partagent le même espace de noms. Par conséquent, vous pouvez utiliser l'instruction **RENAME TABLE** pour renommer une vue.

Voici la syntaxe de base de l'instruction **RENAME TABLE** pour renommer une vue :

```
RENAME TABLE ancien.nom
TO nouveau.nom ;
```

Une autre façon indirecte de renommer une vue consiste à utiliser une séquence des instructions **DROP VIEW** et **CREATE VIEW**.

Tout d'abord, obtenez l'instruction **CREATE VIEW** à l'aide de l'instruction **SHOW CREATE VIEW**.

Ensuite, copiez l'instruction **CREATE VIEW** et enregistrez-la dans un fichier.

Ensuite, supprimez la vue à l'aide de l'instruction **DROP VIEW**.

Ensuite, modifiez le nom de la vue dans l'instruction **CREATE VIEW** dans le fichier.

Enfin, exécutez l'instruction **CREATE VIEW** pour créer la vue avec le nouveau nom.

Notez qu'en utilisant une séquence d'instructions **DROP VIEW** et **CREATE VIEW**, vous pouvez également déplacer une vue d'une base de données à une autre.

SHOW VIEW MySQL

L'instruction **SHOW FULL TABLES** renvoie la liste des vues d'une base de données donnée. Si vous souhaitez afficher l'instruction qui crée la vue, consultez le didacticiel sur l'instruction **SHOW CREATE VIEW**.

La syntaxe :

```
SHOW FULL TABLES WHERE Table_type = 'VIEW';
```

Le résultat en sortie :

Comme l'instruction **SHOW FULL TABLES** renvoie à la fois les tables et les vues, nous devons ajouter une clause **WHERE** pour obtenir uniquement les vues.

Si vous souhaitez afficher toutes les vues d'une base de données spécifique, vous pouvez utiliser la clause **FROM** ou **IN** dans l'instruction **SHOW FULL TABLES** :

```
SHOW FULL TABLES
[{FROM | IN } database_name]
```

```
WHERE Table_type = 'VIEW';
```

Vous indiquez le nom d'une base de données à partir de laquelle vous souhaitez obtenir les vues après la clause **FROM** ou **IN**.

Par exemple, l'instruction suivante affiche toutes les vues de la base de données « ***classicmodels*** » :

```
mysql> SHOW FULL TABLES IN classicmodels WHERE table_type = 'VIEW';
+--------------------------+------------+
| Tables_in_classicmodels | Table_type |
+--------------------------+------------+
| LuxuryItems              | VIEW       |
| aboveAvgProducts         | VIEW       |
| bigSalesOrder            | VIEW       |
| clientPaiements          | VIEW       |
| customerOrderStats       | VIEW       |
| customerOrders           | VIEW       |
| daysofweek               | VIEW       |
| officeInfo               | VIEW       |
| salePerOrder             | VIEW       |
+--------------------------+------------+
9 rows in set (0,00 sec)
```

Si vous souhaitez rechercher les vues qui correspondent à un modèle, vous pouvez utiliser la clause **LIKE** :

```
mysql> SHOW FULL TABLES FROM classicmodels LIKE 'customerOrder%';
+-----------------------------------------+------------+
| Tables_in_classicmodels (customerOrder%) | Table_type |
+-----------------------------------------+------------+
| customerOrderStats                       | VIEW       |
| customerOrders                           | VIEW       |
+-----------------------------------------+------------+
2 rows in set (0,00 sec)
```

> Notez que l'instruction SHOW TABLES ne renvoie que les vues auxquelles vous avez le droit d'accéder.

DROP VIEW MySQL

L'instruction **DROP VIEW** supprime complètement une vue de la base de données. La syntaxe de base de l'instruction **DROP VIEW** :

```
DROP VIEW [IF EXISTS] view_name;
```

Dans cette syntaxe, spécifiez le nom de la vue que vous souhaitez supprimer après les mots-clés **DROP VIEW**. L'option facultative **IF EXISTS** ne supprime la vue que si elle existe.

Pour supprimer plusieurs vues dans une seule instruction :

```
DROP VIEW [IF EXISTS] view1_name, view2_name[,view3_name]...;
```

Si la liste contient une vue qui n'existe pas, l'instruction **DROP VIEW** échoue et ne supprime aucune vue. Toutefois, si vous utilisez l'option **IF EXISTS**, l'instruction **DROP VIEW** génère une **NOTE** pour chaque vue inexistante.