DDL avancé, views et indexs

m2iformation.fr

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Partie 1:

Les indexs

Les indexs

Un index est **une structure de données** utilisée pour **accélérer les recherches dans une table**, un peu comme l'index à la fin d'un livre permet de retrouver rapidement un mot.

Les indexs peuvent être **créées automatiquement par MySQL** (c'est le cas des PRIMARY KEY ou des colonnes présentant la contrainte UNIQUE) ou **manuellement par l'administrateur** de la base de données.

Avantages et inconvénients

Ils présentent des avantages, mais il ne faut pas en abuser.

Avantages:

- **Accélère les opération de lecture**: WHERE, JOIN, ORDER BY, GROUP BY
- Réduit le temps de réponse des requêtes complexes

Inconvénients:

- **Ralentit les opérations d'écriture** (INSERT, UPDATE, DELETE) car l'index doit être mis à jour
- Utilise de l'espace disque supplémentaire

Création d'un index

Il existe deux façons de créer un index, **dès la conception de la table** ou **lors d'une requête ultérieure.**

- Ajout d'un index à la création de la table:

```
CREATE TABLE utilisateurs (
  id INT PRIMARY KEY,
  email VARCHAR(255) UNIQUE,
  nom VARCHAR(100),
  INDEX (nom)
);
```

Création d'un index

- Création d'un index de façon autonome:

```
CREATE INDEX idx_nom ON utilisateurs(nom);
```

- Création d'un **index composite**, c'est à dire qui repose sur une association de données:

```
CREATE INDEX idx_nom_prenom ON utilisateurs(nom, prenom);
```

Afficher les indexs

Comme pour toutes les structures en DDL, vous pouvez afficher les indexs de votre table grâce à la **clause SHOW**

```
SHOW INDEX FROM nom_table;
```

Dans certaines bases plus anciennes, vous trouverez parfois également cette syntaxe (NON RECOMMANDE, moins lisible):

```
SELECT * FROM information_schema.STATISTICS
WHERE table_schema = 'ma_base' AND table_name = 'ma_table';
```

Suppression et modification

On retrouve aussi la **clause DROP** pour la suppression d'un index.

```
DROP INDEX idx_nom ON utilisateurs;
```

ATTENTION, la commande ALTER INDEX **n'existe pas en MySQL.** Contrairement à certains SGBD comme PostgreSQL ou Oracle, MySQL ne permet pas de modifier un index existant directement.
Il faudra donc supprimer puis recréer l'index pour opérer des modifications.

Contraintes de typages

Attention également, **certains types de données ne peuvent pas être indexés.**

C'est le cas notamment des **BLOB** mais aussi des types **TEXT** dont la longueur est indéfinie. C'est pour ça qu'on lui privilégie souvent le type **VARCHAR()** dont la longueur définie permet une indexation par le moteur.

Attention, les stockages très longs (par exemple, VARCHAR(1000)) sont possible à indexer, mais coûteux en espace mémoire et disque. Le moteur peut même refuser selon le longueur totale d'index autorisée

Bonnes pratiques

- Ne pas indexer toutes les colonnes : viser celles fréquemment filtrées ou triées
- Préférer les index composés à plusieurs index simples quand on fait des recherches sur plusieurs colonnes en même temps
- Garder l'œil sur le ratio gain performance / coût en écriture

Cas concret

Situation:

Une table commandes contient 500 000 lignes. On fait souvent cette requête :

```sql
SELECT * FROM commandes WHERE date_commande BETWEEN '2024-01-01' AND '2024-12-31';
```

Optimisation:

Créer un index sur date_commande :

```sql
CREATE INDEX idx_date ON commandes(date_commande);
```

Gestion par le SGBD

MySQL peut ne pas utiliser un index si :

- La table est trop petite (full scan plus rapide)
- L'index est mal conçu (pas sur la bonne colonne)
- L'estimation de coût du moteur estime un full table scan plus optimal

On peut consulter le choix du SGBD par l'utilisation de la clause **EXPLAIN** sur notre requête, nous permettant de voir les indexations inutiles (et de les supprimer). Le type (ref/full scan) nous indique le type de balayage.

Partie 2:

Les views

Introduction

Une vue est une table virtuelle dans MySQL qui permet de sauvegarder des requêtes complexes sous forme d'une table temporaire et d'y accéder facilement.

Une vue ne stocke pas de données proprement dites, elle reflète les résultats d'une requête SQL.

Elles sont souvent utilisées pour simplifier des requêtes complexes, masquer certaines colonnes pour des raisons de sécurité ou encore pour faciliter l'accès aux données par les utilisateurs.

Avantages

- **Simplicité:** Elles permettent de masquer la complexité des requêtes, rendant l'accès aux données plus facile.
- **Sécurité:** Les vues permettent de restreindre l'accès aux colonnes ou lignes spécifiques d'une table, augmentant la confidentialité.
- **Réutilisabilité:** Une vue peut être réutilisée dans plusieurs requêtes sans avoir à écrire plusieurs fois la même requête complexe.
- **Maintenabilité:** Les modifications dans la logique de la requête se font en un seul endroit (la définition de la vue).

Syntaxe de création d'une vue

```
CREATE VIEW nom_vue AS
SELECT colonne1, colonne2, ...
FROM table
WHERE condition;
```

- **CREATE VIEW:** La commande pour créer une vue.
- **nom_vue:** Le nom de la vue qui sera utilisée pour appeler la vue par la suite.
- **SELECT ... FROM ... WHERE:** La requête SQL sur laquelle repose la vue.

Il est également possible de créer une vue à partir d'une autre vue

Conventions de nommage

Utilisez des noms explicites qui reflètent ce qu'elle représente, au singulier, et commencez par vue_, vw_ ou un autre préfixe explicite pour pouvoir les différencier en un coup d'oeil des tables à la lecture de la requête:

- ***vue_clients_actifs***
- ***vw_ventes_par_region***

Utilisation des views

Les views, une fois générées, s'utilisent exactement comme **des tables**.
Elles peuvent d'ailleurs interagir ou fusionner avec d'autres tables concrètes ou d'autres views, comme une table normale.

```
SELECT * FROM vue_clients;
```

Puisque la vue est **dynamique**, toute modification dans les tables sous-jacentes est automatiquement reflétée quand on interroge la vue.

Modifier une view

La commande ALTER VIEW permet de modifier la définition d'une vue existante sans avoir à la supprimer et à la recréer.

```
ALTER VIEW nom_vue AS
SELECT nouvelle_colonne1, nouvelle_colonne2, ...
FROM nouvelle_table
WHERE nouvelle_condition;
```

Remplacer une view

La commande CREATE OR REPLACE VIEW permet de créer une nouvelle vue ou de remplacer une vue existante par une nouvelle définition.

```
CREATE OR REPLACE VIEW nom_vue AS
SELECT colonne1, colonne2, ...
FROM table
WHERE condition;
```

Alter ou Replace ?

1. **Fonctionnalité** :
   - `ALTER VIEW` : Utilisé pour modifier la définition d'une vue existante. Elle ne peut pas être utilisée pour créer une nouvelle vue si celle-ci n'existe pas déjà.
   - `CREATE OR REPLACE VIEW` : Crée une nouvelle vue ou remplace une vue existante. Si la vue n'existe pas, elle la crée ; si elle existe, elle la remplace par la nouvelle définition.

Alter ou Replace ?

2. **Utilisation** :
   - Utilisez `ALTER VIEW` lorsque vous souhaitez effectuer des modifications spécifiques sur une vue sans affecter son existence.
   - Utilisez `CREATE OR REPLACE VIEW` lorsque vous voulez changer complètement la vue ou lorsque vous n'êtes pas sûr de son existence.

Suppression d'une view

La suppression d'une vue se fait à l'aide de la commande DROP VIEW

```
DROP VIEW nom_vue;
```

Attention ! Si vous **supprimez une table dont dépend une vue**, la vue devient évidemment invalide. Toute tentative d'utilisation de la vue renverra une erreur.

Lister les views

Les vues sont stockées dans **le data dictionary** de MySQL, c'est-à-dire dans les métadonnées système.

Elles sont visibles via la commande :

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

Ou via :

```
SELECT * FROM information_schema.VIEWS
WHERE TABLE_SCHEMA = 'nom_de_la_base';
```

Cas concret

Imaginons une base de données pour un magasin en ligne avec trois tables principales : `users` (utilisateurs), `orders` (commandes) et `products` (produits).

On veut créer une vue pour afficher les informations des commandes, y compris le nom de l'utilisateur, le produit acheté, et la quantité commandée.

Tables:

users: contient user_id, name, email

orders: contient order_id, user_id, product_id, quantity

products: contient product_id, product_name, price

Cas concret

Création de la vue :

```
CREATE VIEW orders_view AS
SELECT users.name, products.product_name, orders.quantity
FROM orders
INNER JOIN users ON orders.user_id = users.user_id
INNER JOIN products ON orders.product_id = products.product_id;
```

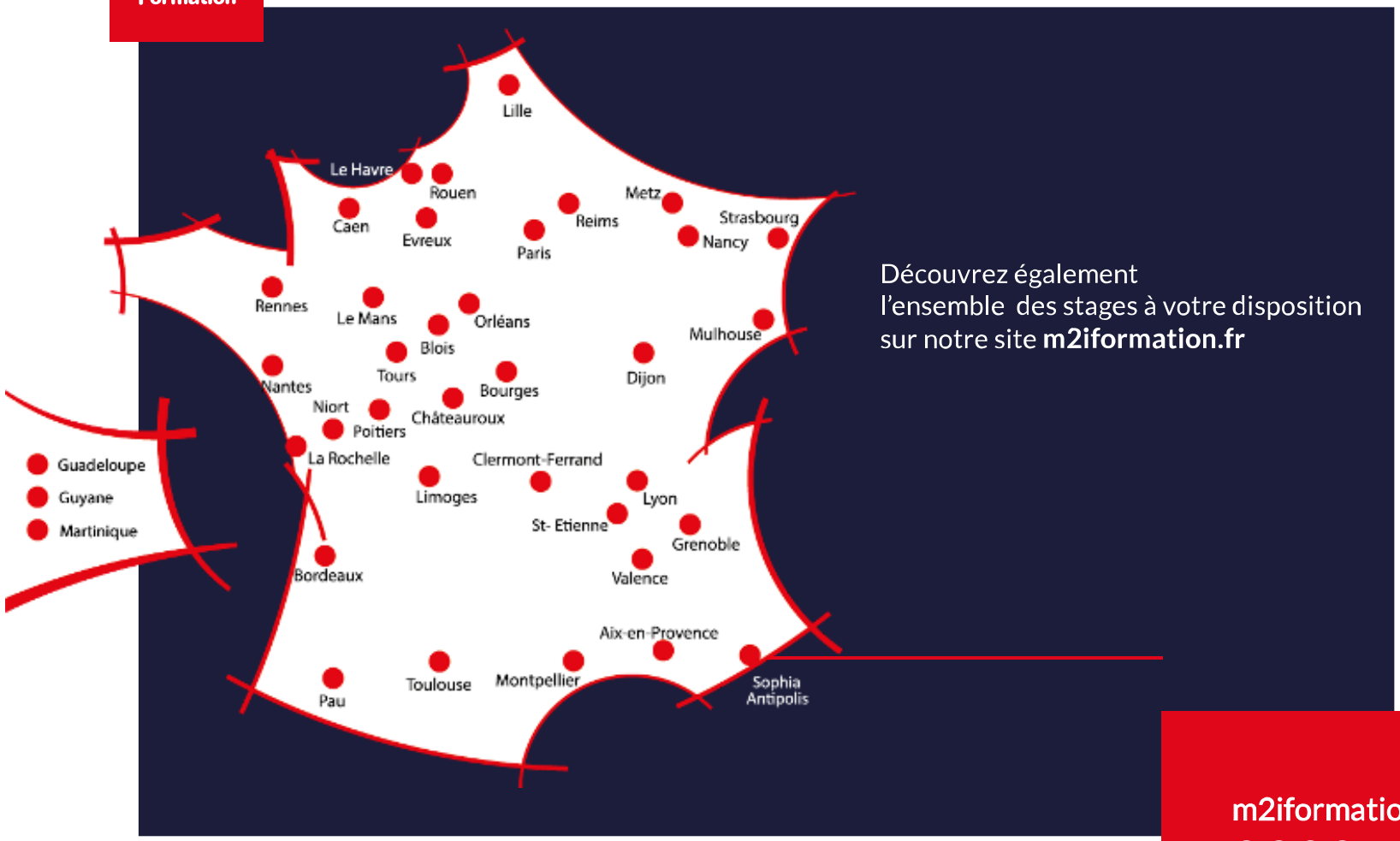
Cette vue permet de simplifier l'accès aux informations des commandes avec le nom de l'utilisateur, le nom du produit et la quantité commandée.

```
SELECT * FROM orders_view;
```

Vues et sécurité

Si vous avez consulté le support sur **le DCL**, vous savez maintenant qu'on peut assigner des droits spéciaux aux utilisateurs.

On peut accorder des droits SELECT sur une vue à un utilisateur, tout en lui refusant l’accès aux tables sous-jacentes. C’est un excellent outil de sécurité.



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr