Labo 4 : Opérations avancées SQL

Objectifs

Dans ce laboratoire, vous allez pratiquer les concepts avancés de SQL, notamment les triggers, les vues et les procédures/fonctions stockées.

Indications

Importez le schéma fourni du labo 3 dans Workbench, pour créer la base de données nécessaire à ce laboratoire (`sakila-schema.sql`), puis importez les données (`sakila-data.sql`) afin de la peupler.
Ce laboratoire est à rendre pour :

- Le 14.12.2015 à 23h55 pour la **classe BDR-1-A-L1**.
- Le 15.12.2015 à 23h55 pour la **classe BDR-1-B-L1**.

Consignes générales

- Le fichier `sakila_schema.pdf` représente la base de données Sakila. Prenez le temps de bien comprendre ce schéma. Le document `sakila.pdf` regroupe toutes les informations concernant cette base de données.
- Vous devez exécuter les requêtes « à la main », c'est-à-dire qu'il est obligatoire d'écrire le code `SQL` de chaque requête, afin de pouvoir l'ajouter au rapport final.
- Tous les mots clés du langage (`SELECT`, `FROM`, `WHERE`, …) doivent être écrit en majuscule, le reste, en minuscule, et les requêtes doivent être **correctement formatées et indentées**.

```sql
SELECT
  first_name,
  last_name,
  email
FROM customer
WHERE email IS NOT NULL
  AND store_id = (
    SELECT
      MAX(store_id)
    FROM store
  )
ORDER BY customer.last_name;
```

- Si le résultat d'une requête fournit un grand nombre de lignes, seules les 20 premières du résultat seront reportées dans le rapport, avec le nombre de lignes total.

titre	nombre_acteurs
LUCKY FLYING	10
OLEANDER CLUE	10
INSIDER ARIZONA	9
WIZARD COLDBLOODED	9
PERSONAL LADYBUGS	9
ALONE TRIP	8
RUNNER MADIGAN	8
CHAMBER ITALIAN	7
TELEGRAPH VOYAGE	7
UNCUT SUICIDES	7
DRIVING POLISH	7
MONSTER SPARTACUS	7
ALASKA PHANTOM	7
HANOVER GALAXY	7
BOOGIE AMELIE	6
HOME PITY	6
CLUE GRAIL	6
TAXI KICK	6
WORDS HUNTER	6
AMELIE HELLFIGHTERS	6

82 17:49:51 51 row(s) returned

Triggers

Vous pouvez vérifier la syntaxe MySQL des triggers ici.
Pour chaque question, montrez que l'opération s'est bien déroulée, c'est-à-dire, qu'il sera nécessaire d'insérer ou de mettre à jour « au moins » une ligne de la base de données, pour afficher le résultat du trigger. Il faudra donc rendre, le code de chaque trigger, le code de l'insertion ou de la mise à jour associée, le code pour afficher la ligne ajoutée ou modifiée, ainsi que des screenshots avant et après la modification appliquée par le trigger.

Attention, il n'est pas possible d'avoir plusieurs triggers avec les mêmes conditions temporelles et événementielles sur la même table, c'est-à-dire qu'il n'est pas possible d'avoir plusieurs triggers `BEFORE INSERT ON` sur la même table, mais vous pouvez avoir plusieurs triggers `BEFORE INSERT ON` sur des tables différentes. Il est toutefois possible d'avoir des triggers en « parallèle » sur la même table, vous allez en général utiliser les 6 combinaisons suivantes :

- `BEFORE INSERT ON`
- `AFTER INSERT ON`
- `BEFORE UPDATE ON`
- `AFTER UPDATE ON`
- `BEFORE DELETE ON`
- `AFTER DELETE ON`

Pour certains triggers demandés, vous allez devoir choisir entre utiliser une autre contrainte temporelle et événementielle, ou faire un `DROP TRIGGER IF EXISTS`, afin de pouvoir créer le nouveau trigger.

1. Utilisez un trigger sur la table `payment`, pour qu'à chaque insertion, le paiement soit majoré de 8%, et que la date du paiement soit mise à jour à la date courante du serveur.
2. Créez :
    a. Une nouvelle table `staff_creation_log` avec les attributs `username` et `when_created`.
    b. Un trigger qui insère une nouvelle ligne dans `staff_creation_log`, à chaque fois qu'une nouvelle entrée est insérée dans la table `staff`.
3. Utilisez un trigger permettant de mettre à jour automatiquement, lors d'une insertion, l'adresse e-mail d'un membre du `staff`, sous la forme `prénom.nom@sakilastaff.com`.
4. Créez :
    a. Une nouvelle table `customer_store_log` avec les attributs `customer_id`, `last_store_id`, `register_date` et `unregister_date` dont le but est d'archiver les anciens magasins fréquentés par un client. `register_date` représente la date d'inscription d'un client dans un magasin et `unregister_date` représente la date de sa désinscription.
    b. Un trigger pour enregistrer une nouvelle ligne dans la table `customer_store_log`, à chaque fois qu'un client change de magasin.

Event

La documentation relative aux événements MySQL se trouve ici.
Il ne faut pas oublier d'ajouter la ligne `SET GLOBAL event_scheduler = ON/OFF`, pour démarrer, respectivement arrêter, le service de gestions des événements MySQL.

5. Créer un événement qui permet de nettoyer une fois par minute la table `customer_store_log`, afin de supprimer les enregistrements des clients qui ont changé au moins deux fois de magasin, et dont leur changement le plus récent, date de plus d'une année.

Vues

Vous pouvez vérifier la syntaxe MySQL des vues ici.
Pour chaque question, montrez que l'opération s'est bien déroulée. Il faudra donc rendre, le code de chaque vue, le code pour afficher les 20 premières lignes, ainsi que un screenshot du résultat obtenu.

6. On aimerait envoyer des cartes du Nouvel An à l'adresse postale du personnel, et on souhaiterait déléguer cette tâche à un employé, Franklin, mais pour une question de protection des données personnelles, il ne doit avoir accès qu'à certaines données. Il a besoin du numéro de téléphone ainsi que de l'adresse postale, mais en aucun cas l'adresse e-mail ou encore le mot de passe.
    a. Créez une vue sur la table du personnel qui permettra à Franklin de réaliser la tâche demandée.
    b. Est-ce que Franklin pourra mettre à jour la base de donnée à travers cette vue ?
7. On aimerait demander à Franklin d'envoyer un rappel par e-mail à tous les clients qui ont du retard. Créez la vue dont il a besoin pour effectuer cette tâche. L'e-mail envoyé à chaque client doit contenir le titre du film qu'ils n'ont pas rendu, ainsi que le nombre de jours de retard.
8. Utilisez la vue créée en 7, pour afficher le nombre de clients ayant plus de 3 jours de retard.
9. On aimerait demander à Franklin de calculer le nombre de location par client.
    a. Créez la vue dont il a besoin pour effectuer cette tâche.
    b. Affichez les clients ayant le plus de location.
10. Franklin souhaite faire quelques statistiques.
    a. Créez une vue pour calculer le nombre total de location par jour.
    b. Combien de locations ont été effectuées le 1er août 2005 ?

Fonctions et procédures stockées

Vous pouvez vérifier la syntaxe MySQL des fonctions et procédures stockées ici.
Pour chaque question, montrez que l'opération s'est bien déroulée. Il faudra donc rendre, le code de chaque fonction ou procédure, le code qui appelle la fonction ou procédure, le code pour afficher le résultat sans utiliser la fonction ou la procédure, ainsi que un screenshot du résultat obtenu, avec comme limite, les 20 premières lignes.

11. Créez une fonction qui calcule le nombre de films proposés par un magasin.
    a. La fonction prend en entrée l'`id` du magasin et retourne en sortie, le nombre de films proposés par ce magasin.
    b. Utilisez la fonction pour afficher le nombre de films proposés par les magasins `1` et `2`.
12. Créez une fonction qui calcule le nombre de clients par magasin.
    a. La fonction prend en entrée l'`id` du magasin et retourne en sortie, le nombre de clients pour ce magasin.
    b. Utilisez la fonction pour afficher le nombre de clients pour les magasins `1` et `2`.
13. Créez une fonction qui calcule le revenu de chaque magasin.
    a. La fonction prend en entrée l'`id` du magasin et retourne en sortie, le revenu total du magasin.
    b. Utilisez la fonction pour afficher le revenu des magasins `1` et `2`.
14.
    a. Créez une procédure pour mettre à jour l'attribut `last_update` de tous les films, à la date d'exécution de la procédure.
    b. Quelle est la date de mise à jour des films avant et après cette procédure ?
15.
    a. Créez une procédure qui calcule le nombre d'exemplaires physiques des films, ainsi que le nombre de locations dans chaque magasin.
    b. Utilisez la procédure pour afficher le résultat des magasins `1` et `2`.