Lab 6: Apache Hive

L'objectif de ce TP est de :

- installation d'apache Hive
- Première utilisation d'apache Hive
- Réaliser des requêtes analytiques

Apache Hive est un software **datawarehouse** conçu pour lire, écrire et gérer de grands ensembles de données extraits du système de fichiers distribué d'Apache Hadoop (HDFS).
En effet, il ne s'agit pas d'une base de données complète. Il ne stocke que les métadonnées et les données sont stockées uniquement dans HDFS. Ainsi, chaque requête écrite par l'utilisateur est convertie en code MapReduce qui interagit ensuite avec HDFS. Hive peut être utilisé comme système OLAP (Online Analytical Processing).

Hive est fourni avec HiveServer2, une interface serveur dotée de sa propre interface de ligne de commande (CLI) appelée **Beeline (client JDBC)**, qui permet de se connecter à Hive,

I. Installation Apache Hive

- Pull l'image depuis dockerHub : https://hub.docker.com/r/apache/hive/tags. La dernière image est 4.0.0-alpha-2

```
docker pull apache/hive:4.0.0-alpha-2
```

Pour une configuration rapide, démarrer HiveServer2 avec le Metastore « embedded »
(SGBD Derby comme BD metastore)

- Lancer ensuite les démons yarn et hdfs:

```
docker run -v ~/Documents/hadoop_project/:/shared_volume -d -p
10000:10000 -p 10002:10002 -p 9083:9083 --env SERVICE_NAME=hiveserver2 --
name hiveserver2-standalone apache/hive:4.0.0-alpha-2
```

- Pour accéder à HiveServer2 web via le navigateur via `http://localhost:10002`

II. Première utilisation beeline

Pour se connecter à serveur hive **hiveserver2**, on peut utiliser à l'invité de commande **Beeline** en fournissant une adresse IP et le port sur la chaîne d'URL de connexion JDBC. (Par défaut, le serveur écoute sur le port=10000)

- Accéder au shell du conteneur hiveserver2-standalone

```
docker exec -it hiveserver2-standalone bash
```

- HDFS est souvent déjà démarré dans les configurations Docker par défaut. Vérifier que HDFS est opérationnel :

```
hadoop fs -ls
```

- visualiser le contenu du fichier de configuration ***hive-site.xml*** dans ***/opt/hive/conf/***

- Accéder au shell de hive beeline en utilisant la commande suivante : login « **scott** » avec mot de passe « ***tiger*** »)

```
beeline -u jdbc:hive2://localhost:10000 scott tiger
```

- afficher les bases de données disponibles:

```
  show databases ;
```

III. Analyse de données de réservation d'hôtels

Dans ce TP, nous allons travailler sur un ensemble de données concernant les réservations d'hôtels. L'objectif principal est de manipuler, analyser et extraire des informations pertinentes sur les clients, les hôtels et leurs réservations. Pour ce faire, nous utiliserons Apache Hive pour stocker et interroger les données. Les données sont disponibles dans trois fichiers :

1. Créer la base de données

- Créer la base de données hotel_booking ;

```
CREATE DATABASE hotel_booking;
USE hotel_booking;
```

- Lister le contenu du répertoire /opt/hive/data/warehouse. Qu'est ce que vous remarquez ?

2. Créer les tables

- Créer les tables pour stocker les informations des clients et des hôtels.
- Pour employer les partitions et les buckets, il faudra commencer par activer les propriétés suivantes :

```
set hive.exec.dynamic.partition=true ;
set hive.exec.dynamic.partition.mode=nonstrict ;
set hive.exec.max.dynamic.partitions=20000 ;
set hive.exec.max.dynamic.partitions.pernode=20000 ;
set hive.enforce.bucketing = true ;
```

```
CREATE TABLE clients ( client_id INT, nom STRING, email STRING,
telephone STRING )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Créer la table ***reservations*** avec une partition par ***date_debut*** pour optimiser les requêtes en fonction des dates.

3. Charger les données dans les tables

- Charger les données dans la table clients et hotels.

```
LOAD DATA LOCAL INPATH '/path/to/clients.txt' INTO TABLE clients;
```

- Charger les données dans les tables reservations et hotels.

```
LOAD DATA LOCAL INPATH '/path/to/reservations.txt' INTO TABLE
reservations PARTITION (date_debut);
```

4. Utiliser des partitions et des buckets

- Nous allons modifier la table ***hotels*** pour ajouter des partitions par ***ville*** et utiliser des buckets pour la table des réservations par ***client_id*** afin d'optimiser les jointures.

```
CREATE TABLE hotels_partitioned (
    hotel_id INT,
    nom STRING,
    etoiles INT
)
PARTITIONED BY (ville STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

```
CREATE TABLE reservations_bucketed (
    reservation_id INT,
    client_id INT,
    hotel_id INT,
    date_debut DATE,
    date_fin DATE,
    prix_total DECIMAL(10,2)
)
CLUSTERED BY (client_id) INTO 4 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

- Charger les tables créés.
- Lister le contenu du répertoire /opt/hive/data/warehouse/hotel_booking.db/. Qu'est ce que vous remarquez ?

5. Utilisation de requêtes simples

- Lister tous les clients
- Lister tous les hôtels à Paris
- Lister toutes les réservations avec les informations sur les hôtels et les clients

6. Requêtes avec jointures

- Afficher le nombre de réservations par client
- Afficher les clients qui ont réservé plus que 2 nuitées
- Afficher les Hôtels réservés par chaque client
- Afficher les noms des hôtels dans lesquels il y a plus qu'une réservation.
- Afficher les noms des hôtels dans lesquels il y a pas de réservation.

7. Requêtes imbriquées

- Afficher les clients ayant réservé un hôtel avec plus de 4 étoiles
- Afficher le Total des revenus générés par chaque hôtel

8. Utilisation de fonctions d'agrégation avec partitions et buckets

- Revenus totaux par ville (partitionnée)
- Nombre total de réservations par client (bucketed)

9. Nettoyage et suppression des données

Supprimer les tables créés précédemment.

10. Script hql

Refaire le même traitement en organisant le code en trois scripts HiveQL distincts :

1. **Creation.hql** : Contient les commandes pour créer la base de données, les tables (clients, hôtels, réservations), ainsi que les partitions et les buckets si nécessaires.
2. **Loading.hql** : Contient les instructions pour charger les données depuis les fichiers texte (clients.txt, hotels.txt, reservations.txt) dans les tables Hive.
3. **Queries.hql** : Inclut toutes les requêtes SQL précédentes, telles que les jointures, les agrégations et les requêtes imbriquées.

Ces scripts doivent être exécutés dans l'ordre pour reproduire entièrement le traitement