eBook Gratuit

APPRENEZ hive

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#hive

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: hive

It is an unofficial and free hive ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official hive.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec la ruche

Remarques

- Hive est un outil d'entrepôt de données construit sur Hadoop .
- Il fournit un langage de type SQL pour interroger les données.
- Nous pouvons exécuter presque toutes les requêtes SQL dans Hive, à la seule différence que cela exécute un travail de réduction de carte au niveau du serveur principal pour récupérer le résultat du cluster Hadoop. À cause de cela, Hive prend parfois plus de temps pour récupérer le jeu de résultats.

Examples

Exemple de compte de mots dans Hive

Fichier de documentation (fichier d'entrée)

Mary avait un petit agneau

sa toison était blanche comme neige

et partout où Marie est allée

l'agneau était sûr d'y aller.

Requête Ruche

```
CREATE TABLE FILES (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE FILES;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, ' ')) AS word FROM FILES) w
GROUP BY word
ORDER BY word;
```

Sortie de la table word_counts dans Hive

Marie, 2 ans

eu, 1

un, 1

peu 1

agneau, 2

sa, 1

polaire, 1

était, 2

blanc, 1

comme 1

neige, 1

et 1

partout, 1

ça, 1

est allé, 1

le, 1

bien sûr, 1

à, 1

allez, 1

Installation de Hive (linux)

Commencez par télécharger la dernière version stable depuis
https://hive.apache.org/downloads.html

-> Maintenant, décompressez le fichier avec

$ tar -xvf hive-2.xy-bin.tar.gz

-> Créez un répertoire dans / usr / local / with

$ sudo mkdir / usr / local / hive

-> Déplacer le fichier avec root

$ mv ~ / Téléchargements / hive-2.xy / usr / local / ruche

-> Modifier les variables d'environnement pour hadoop et ruche dans .bashrc

$ gedit ~ / .bashrc

comme ça

export HIVE_HOME = / usr / local / hive / apache-hive-2.0.1-bin /

export PATH = $ PATH: $ HIVE_HOME / bin

export CLASSPATH = $ CLASSPATH: / usr / local / Hadoop / lib / * :.

export CLASSPATH = $ CLASSPATH: /usr/local/hive/apache-hive-2.0.1-bin/lib/* :.

-> Maintenant, démarrez hadoop si ce n'est pas déjà fait. Et assurez-vous qu'il fonctionne et qu'il n'est pas en mode sans échec.

$ hadoop fs -mkdir / user / hive / warehouse

Le répertoire "warehouse" est l'emplacement où stocker la table ou les données liées à la ruche.

$ hadoop fs -mkdir / tmp

Le répertoire temporaire «tmp» est l'emplacement temporaire pour stocker le résultat intermédiaire du traitement.

-> Définir les autorisations pour lire / écrire sur ces dossiers.

$ hadoop fs -chmod g + w / user / ruche / entrepôt

$ hadoop fs -chmod g + w / user / tmp

-> Maintenant lancez HIVE avec cette commande dans la console

$ ruche

Installation de ruche avec métastore externe sous Linux

Conditions préalables:

1. Java 7
2. Hadoop (Voir ici pour l'installation de Hadoop)
3. Mysql Server et Client

Installation:

Étape 1: Téléchargez la dernière archive tar Hive à partir de la page des téléchargements .

Étape 2: Extraire l'archive tar téléchargée (***Hypothèse:*** *l'archive tar est téléchargée dans $ HOME*)

```
tar -xvf /home/username/apache-hive-x.y.z-bin.tar.gz
```

Étape 3: Mettez à jour le fichier d'environnement (`~/.bashrc`)

```
export HIVE_HOME=/home/username/apache-hive-x.y.z-bin
export PATH=$HIVE_HOME/bin:$PATH
```

source du fichier pour définir les nouvelles variables d'environnement.

```
source ~/.bashrc
```

Étape 4: Télécharger le connecteur JDBC pour MySQL à partir d' ici et l' extraire.

```
tar -xvf mysql-connector-java-a.b.c.tar.gz
```

Le répertoire extrait contient le fichier jar de connecteur `mysql-connector-java-abcjar` . Copiez-le dans la `lib` de `$HIVE_HOME`

```
cp mysql-connector-java-a.b.c.jar $HIVE_HOME/lib/
```

Configuration:

Créez le fichier de configuration de `hive-site.xml` sous le `hive-site.xml $HIVE_HOME/conf/` et ajoutez les propriétés associées au métastore suivantes.

```
<configuration>
   <property>
      <name>javax.jdo.option.ConnectionURL</name>
      <value>jdbc:mysql://localhost/hive_meta</value>
      <description>JDBC connect string for a JDBC metastore</description>
   </property>

   <property>
      <name>javax.jdo.option.ConnectionDriverName</name>
      <value>com.mysql.jdbc.Driver</value>
      <description>Driver class name for a JDBC metastore</description>
   </property>

   <property>
       <name>javax.jdo.option.ConnectionUserName</name>
       <value>mysqluser</value>
       <description>username to use against metastore database</description>
   </property>

   <property>
       <name>javax.jdo.option.ConnectionPassword</name>
       <value>mysqlpass</value>
       <description>password to use against metastore database</description>
   </property>

   <property>
       <name>datanucleus.autoCreateSchema</name>
       <value>false</value>
   </property>

   <property>
       <name>datanucleus.fixedDatastore</name>
       <value>true</value>
   </property>
</configuration>
```

Mettez à jour les valeurs de MySQL "username" et "password" en conséquence dans les propriétés.

Créez le schéma Metastore:

Les scripts `$HIVE_HOME/scripts/metastore/upgrade/mysql/` sont disponibles sous `$HIVE_HOME/scripts/metastore/upgrade/mysql/`

Connectez-vous à MySQL et fournissez le schéma,

```
mysql -u username -ppassword

mysql> create database hive_meta;
mysql> use hive_meta;
mysql> source hive-schema-x.y.z.mysql.sql;
mysql> exit;
```

Démarrer Metastore:

```
hive --service metastore
```

Pour l'exécuter en arrière-plan,

```
nohup hive --service metastore &
```

Démarrer HiveServer2: (à utiliser si nécessaire)

```
hiveserver2
```

Pour l'exécuter en arrière-plan,

```
nohup hiveserver2 metastore &
```

Remarque: Ces exécutables sont disponibles sous `$HIVE_HOME/bin/`

Relier:

Utilisez soit `hive` , `beeline` ou Hue pour vous connecter avec Hive.

Hive CLI est obsolète, l'utilisation de Beeline ou de Hue est recommandée.

Configurations supplémentaires pour Hue:

Mettez à jour cette valeur dans `$HUE_HOME/desktop/conf/hue.ini`

```
[beeswax]
    hive_conf_dir=/home/username/apache-hive-x.y.z-bin/conf
```

Lire Commencer avec la ruche en ligne: https://riptutorial.com/fr/hive/topic/1099/commencer-avec-la-ruche

Chapitre 2: Création de table de ruche à travers Sqoop

Introduction

Si nous avons un méta-magasin Hive associé à notre cluster HDFS, Sqoop peut importer les données dans Hive en générant et en exécutant une instruction CREATE TABLE pour définir la disposition des données dans Hive. Importer des données dans Hive est aussi simple que d'ajouter l'option --hive-import à votre ligne de commande Sqoop.

Remarques

L'importation de données directement du SGBDR vers HIVE peut permettre de résoudre de nombreux problèmes. Nous pouvons également exécuter une requête de forme libre (une jointure ou une requête simple) et la remplir directement dans une table de notre choix dans Hive.

> --hive-import dit à Sqoop que la destination finale est Hive et non HDFS.
>
> L'option --hive-table aide à importer les données dans la table de hive choisie par nous, sinon elle sera nommée en tant que table source importée à partir du SGBDR.

Examples

Import de ruche avec le nom de la table de destination dans la ruche

```
$ sqoop import --connect jdbc:mysql://10.0.0.100/hadooptest
--username hadoopuser -P
--table table_name --hive-import --hive-table hive_table_name
```

Lire Création de table de ruche à travers Sqoop en ligne:
https://riptutorial.com/fr/hive/topic/10685/creation-de-table-de-ruche-a-travers-sqoop

Chapitre 3: Créer une déclaration de base de données et de table

Syntaxe

- CREATE [TEMPORARY] [EXTERNAL] TABLE [SI PAS EXISTE] [nom_base.] Nom_table

  [(nom_colonne type_données [COMMENT col_comment], ...)] [COMMENTER table_comment] [PARTITIONED BY (nom_colonne type_données [COMMENT col_comment], ...)] [CLUSTERED BY (nom_colonne, nom_colonne, ...) [SORTED BY (nom_col [ASC | DESC], ...) INTO num_buckets BUCKETS] [SKEWED BY (nom_colonne, nom_colonne, ...) - (Remarque: Disponible dans Hive 0.10.0 et versions ultérieures)] ON ((col_value, col_value, ...), (col_value, col_value, ...), ...) [STOCKÉ COMME RÉPERTOIRES] [[ROW FORMAT row_format] [STORED AS file_format] | STOCKÉ PAR 'storage.handler.class.name' [AVEC SERDEPROPERTIES (...)]] [LOCATION hdfs_path] [TBLPROPERTIES (property_name = value_value, ...)]
  [AS select_statement];

- CREATE [TEMPORARY] [EXTERNAL] TABLE [SI N'EXISTE PAS] [nom_base.] Nom_table LIKE existing_table_or_view_name [LOCATION hdfs_path];

- data_type: primitive_type, array_type, map_type, struct_type, union_type

- primitive_type: TINYINT, SMALLINT, INT, BIGINT, BOOLEAN, FLOAT, DOUBLE, STRING, BINARY, TIMESTAMP, DECIMAL, DECIMAL (précision, échelle), DATE, VARCHAR, CHAR

- array_type: ARRAY <data_type>

- map_type: MAP <primitive_type, data_type>

- struct_type: STRUCT <nom_colonne: type_données [commentaire col_comment], ...>

- union_type: UNIONTYPE <data_type, data_type, ...>

- row_format: DELIMITED [FIELDS TERMINATED BY char [Caractères ESCAPED BY]] [ELEMENTS DE COLLECTION TERMINES PAR char] [MAP KEYS TERMINATED BY char] [LIGNES TERMINÉES PAR char] [NULL DEFINED AS char]
  , SERDE nom_de_serveur [AVEC SERDEPROPERTIES (nom_propriété = valeur_propriété, nom_propriété = valeur_propriété, ...)]

- format_fichier:: SEQUENCEFILE, TEXTFILE, RCFILE, ORC, PARQUET, AVRO, INPUTFORMAT nom_classe_format d'entrée OUTPUTFORMAT nom_classe_format

- CREATE (DATABASE | SCHEMA) [IF NOT EXISTS] nom_de_base_de_données [COMMENT_commentation_base] [LOCATION hdfs_path] [WITH DBPROPERTIES (nom_propriété = valeur_propriété, ...)];

Remarques

Lorsque vous travaillez avec des tables et des bases de données dans HIVE. Les points ci-dessous peuvent être utiles.

- Nous pouvons changer de base de données en utilisant la `use database;` commander
- Pour connaître la base de données de travail actuelle, nous pouvons utiliser `SELECT current_database()`
- Pour voir la DDL utilisée pour l'instruction create table, nous pouvons utiliser `SHOW CREATE TABLE tablename`
- Pour voir toutes les colonnes de la table, utilisez `DESCRIBE tablename` pour afficher des détails étendus tels que location serde used et autres `DESCRIBE FORMATTED tablename` . DESCRIBE peut également être abrégé en DESC.

Examples

Créer une table

Création d'une table **gérée** avec partition et stockée en tant que fichier de séquence. Le format de données dans les fichiers est supposé être délimité par des champs par `Ctrl-A (^A)` et délimité par des lignes sur newline. Le tableau ci-dessous est créé dans le répertoire de stockage de ruche spécifié en valeur pour la clé `hive.metastore.warehouse.dir` dans le fichier de configuration Hive `hive-site.xml` .

```
CREATE TABLE view
(time INT,
id BIGINT,
url STRING,
referrer_url STRING,
add STRING COMMENT 'IP of the User')
COMMENT 'This is view table'
PARTITIONED BY(date STRING, region STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
STORED AS SEQUENCEFILE;
```

Création d'une table **externe** avec des partitions et stockée en tant que fichier de séquence. Le format de données dans les fichiers est supposé être délimité par des champs par `ctrl-A` et délimité par des lignes sur newline. Le tableau ci-dessous est créé à l'emplacement spécifié et il est pratique lorsque nous avons déjà des données. L'un des avantages de l'utilisation d'une table externe est que nous pouvons supprimer la table sans supprimer les données. Par exemple, si nous créons une table et réalisons que le schéma est faux, nous pouvons sans risque déposer la table et recréer avec le nouveau schéma sans nous soucier des données. Un autre avantage est que si nous utilisons d'autres outils comme nous pouvons continuer à les utiliser même après avoir supprimé la table.

```
CREATE EXTERNAL TABLE view
(time INT,
id BIGINT,
```

```
url STRING,
referrer_url STRING,
add STRING COMMENT 'IP of the User')
COMMENT 'This is view table'
PARTITIONED BY(date STRING, region STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
STORED AS SEQUENCEFILE
LOCATION '<hdfs_location>';
```

En créant une table à l'aide de select query et en renseignant les résultats de la requête, ces instructions sont appelées **CTAS (Create Table As Select)** .

Il y a deux parties dans CTAS, la partie SELECT peut être n'importe quelle instruction SELECT prise en charge par HiveQL. La partie CREATE du CTAS prend le schéma résultant de la partie SELECT et crée la table cible avec d'autres propriétés de table telles que SerDe et le format de stockage.

CTAS a ces restrictions:

- La table cible ne peut pas être une table partitionnée.
- La table cible ne peut pas être une table externe.
- La table cible ne peut pas être une table de classement de liste.

```
CREATE TABLE new_key_value_store
ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"
STORED AS RCFile
AS
SELECT * FROM page_view
SORT BY url, add;
```

Créer un tableau comme:

La **forme LIKE de CREATE TABLE** vous permet de copier une définition de table existante exactement (sans copier ses données). Contrairement à CTAS, l'instruction ci-dessous crée une nouvelle table dont la définition correspond exactement à la table existante dans tous les détails autres que le nom de la table. La nouvelle table ne contient aucune ligne.

```
CREATE TABLE empty_page_views
LIKE page_views;
```

Créer une base de données

Créer une base de données dans un emplacement particulier. Si nous ne spécifions aucun emplacement pour la base de données, il est créé dans le répertoire du magasin.

```
CREATE DATABASE IF NOT EXISTS db_name
COMMENT 'TEST DATABASE'
LOCATION /PATH/HDFS/DATABASE/;
```

Hive Création de table ACID.

Les tables ACID sont supportées depuis la version ruche 0.14. Le tableau ci-dessous prend en charge UPDATE / DELETE / INSERT

Les modifications de configuration ci-dessous sont requises dans hive-site.xml.

```
hive.support.concurrency = true
hive.enforce.bucketing = true
hive.exec.dynamic.partition.mode = nonstrict
hive.txn.manager =org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
hive.compactor.initiator.on = true
hive.compactor.worker.threads = 1
```

Actuellement, seul le fichier orc est pris en charge par le format.

Table create statement.

```
create table Sample_Table(
col1 Int,
col2 String,
col3 String)
clustered by (col3) into 3 buckets
stored as orc
TBLPROPERTIES ('transactional'='true');
```

Intégration HIVE_HBASE

L'intégration Hive-Hbase est prise en charge depuis les versions inférieures. Hive: 0.11.0 HBase: 0.94.2 Hadoop: 0.20.2

```
CREATE TABLE hbase_hive
(id string,
 col1 string,
 col2 string,
 col3 int)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES
("hbase.columns.mapping" = ":key,cf1:col1,cf1:col2,cf1:col3")
TBLPROPERTIES ("hbase.table.name" = "hive_hbase");
```

Remarque: la première colonne doit être la colonne clé.

Créez une table à l'aide des propriétés de table existantes.

```
CREATE TABLE new_table_name LIKE existing_table_name;
```

Lire Créer une déclaration de base de données et de table en ligne: https://riptutorial.com/fr/hive/topic/3328/creer-une-declaration-de-base-de-donnees-et-de-table

Chapitre 4: Exporter des données dans la ruche

Examples

Fonction d'exportation dans la ruche

Exportation de données de la table des employés vers / tmp / ca_employees

INSERT OVERWRITE LOCAL DIRECTORY '/ tmp / ca_employees' nom SELECT, salaire, adresse FROM employee WHERE se.state = 'CA';

Exportation de données de la table des employés vers plusieurs répertoires locaux en fonction de conditions spécifiques

La requête ci-dessous montre comment une seule construction peut être utilisée pour exporter des données vers plusieurs répertoires en fonction de critères spécifiques.

```
FROM employees se INSERT OVERWRITE DIRECTORY '/tmp/or_employees' SELECT * WHERE se.cty = 'US'
and se.st = 'OR'
INSERT OVERWRITE DIRECTORY '/tmp/ca_employees' SELECT * WHERE se.cty = 'US' and se.st = 'CA'
INSERT OVERWRITE DIRECTORY '/tmp/il_employees' SELECT * WHERE se.cty = 'US' and se.st = 'IL';
```

Lire Exporter des données dans la ruche en ligne:
https://riptutorial.com/fr/hive/topic/6530/exporter-des-donnees-dans-la-ruche

Chapitre 5: Fonctions d'agrégat définies par l'utilisateur (UDAF)

Examples

UDAF signifie exemple

- Créez une classe Java qui étend `org.apache.hadoop.hive.ql.exec.hive.UDAF` Créez une classe interne qui implémente `UDAFEvaluator`
- Mettre en œuvre cinq méthodes
    - `init()` - Cette méthode initialise l'évaluateur et réinitialise son état interne. Nous utilisons new Column () dans le code ci-dessous pour indiquer qu'aucune valeur n'a encore été agrégée.
    - `iterate()` - Cette méthode est appelée chaque fois qu'il y a une nouvelle valeur à agréger. L'évaluateur doit mettre à jour son état interne avec le résultat de l'agrégation (nous faisons la somme - voir ci-dessous). Nous retournons true pour indiquer que l'entrée était valide.
    - `terminatePartial()` - Cette méthode est appelée lorsque Hive souhaite un résultat pour l'agrégation partielle. La méthode doit renvoyer un objet qui encapsule l'état de l'agrégation.
    - `merge()` - Cette méthode est appelée lorsque Hive décide de combiner une agrégation partielle avec une autre.
    - `terminate()` - Cette méthode est appelée lorsque le résultat final de l'agrégation est requis.

```
public class MeanUDAF extends UDAF {
// Define Logging
static final Log LOG = LogFactory.getLog(MeanUDAF.class.getName());
public static class MeanUDAFEvaluator implements UDAFEvaluator {
/**
 * Use Column class to serialize intermediate computation
 * This is our groupByColumn
 */
public static class Column {
 double sum = 0;
 int count = 0;
 }
private Column col = null;
public MeanUDAFEvaluator() {
 super();
 init();
 }
// A - Initalize evaluator - indicating that no values have been
// aggregated yet.
public void init() {
 LOG.debug("Initialize evaluator");
 col = new Column();
```

```
 }
// B- Iterate every time there is a new value to be aggregated
 public boolean iterate(double value) throws HiveException {
 LOG.debug("Iterating over each value for aggregation");
 if (col == null)
 throw new HiveException("Item is not initialized");
 col.sum = col.sum + value;
 col.count = col.count + 1;
 return true;
 }
// C - Called when Hive wants partially aggregated results.
 public Column terminatePartial() {
 LOG.debug("Return partially aggregated results");
 return col;
 }
 // D - Called when Hive decides to combine one partial aggregation with another
 public boolean merge(Column other) {
 LOG.debug("merging by combining partial aggregation");
 if(other == null) {
 return true;
 }
 col.sum += other.sum;
 col.count += other.count;
 return true;
}
 // E - Called when the final result of the aggregation needed.
 public double terminate(){
 LOG.debug("At the end of last record of the group - returning final result");
 return col.sum/col.count;
 }
 }
}


hive> CREATE TEMPORARY FUNCTION <FUNCTION NAME> AS 'JAR PATH.jar';
hive> select id, mean_udf(amount) from table group by id;
```

Lire Fonctions d'agrégat définies par l'utilisateur (UDAF) en ligne:
https://riptutorial.com/fr/hive/topic/5137/fonctions-d-agregat-definies-par-l-utilisateur--udaf-

Chapitre 6: Fonctions de table définies par l'utilisateur (UDTF)

Examples

Exemple UDTF et utilisation

Fonctions de table définies par l'utilisateur représentées par l'interface **org.apache.hadoop.hive.ql.udf.generic.GenericUDTF** . Cette fonction permet de générer plusieurs lignes et plusieurs colonnes pour une seule entrée.

Nous devons remplacer les méthodes ci-dessous:

```
 1.we specify input and output parameters
abstract StructObjectInspector initialize(ObjectInspector[] args)
                                                throws UDFArgumentException;

 2.we process an input record and write out any resulting records
abstract void process(Object[] record) throws HiveException;

3.function is Called to notify the UDTF that there are no more rows to process.
    Clean up code or additional output can be produced here.
abstract void close() throws HiveException;




import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import org.apache.hadoop.hive.serde2.objectinspector.PrimitiveObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory;

public class NameParserGenericUDTF extends GenericUDTF {
      private PrimitiveObjectInspector stringOI = null;

       //Defining input argument as string.
      @Override
      public StructObjectInspector initialize(ObjectInspector[] args) throws
UDFArgumentException {
        if (args.length != 1) {
          throw new UDFArgumentException("NameParserGenericUDTF() takes exactly one
argument");
        }

        if (args[0].getCategory() != ObjectInspector.Category.PRIMITIVE
```

```
            && ((PrimitiveObjectInspector) args[0]).getPrimitiveCategory() !=
PrimitiveObjectInspector.PrimitiveCategory.STRING) {
          throw new UDFArgumentException("NameParserGenericUDTF() takes a string as a
parameter");
        }

        // input
        stringOI = (PrimitiveObjectInspector) args[0];

        // output
        List<String> fieldNames = new ArrayList<String>(2);
        List<ObjectInspector> fieldOIs = new ArrayList<ObjectInspector>(2);
        fieldNames.add("name");
        fieldNames.add("surname");
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        return ObjectInspectorFactory.getStandardStructObjectInspector(fieldNames, fieldOIs);
      }

      public ArrayList<Object[]> processInputRecord(String name){
            ArrayList<Object[]> result = new ArrayList<Object[]>();

            // ignoring null or empty input
            if (name == null || name.isEmpty()) {
              return result;
            }

            String[] tokens = name.split("\\s+");

            if (tokens.length == 2){
                result.add(new Object[] { tokens[0], tokens[1] });
            }else if (tokens.length == 4 && tokens[1].equals("and")){
                result.add(new Object[] { tokens[0], tokens[3] });
                result.add(new Object[] { tokens[2], tokens[3] });
            }

            return result;
      }

      @Override
      public void process(Object[] record) throws HiveException {
        final String name = stringOI.getPrimitiveJavaObject(record[0]).toString();
        ArrayList<Object[]> results = processInputRecord(name);

        Iterator<Object[]> it = results.iterator();

        while (it.hasNext()){
            Object[] r = it.next();
            forward(r);
        }
      }

      @Override
      public void close() throws HiveException {
        // do nothing
      }
}
```

Emballez le code dans jar et devez ajouter jar au contexte de la ruche.

```
hive> CREATE TEMPORARY FUNCTION process_names as 'jar.path.NameParserGenericUDTF';

Here we will pass input as full name and break it into first and last name.

hive> SELECT
   t.name,
   t.surname
FROM people
    lateral view process_names(name) t as name, surname;

Teena Carter
John Brownewr
```

Lire Fonctions de table définies par l'utilisateur (UDTF) en ligne: https://riptutorial.com/fr/hive/topic/6502/fonctions-de-table-definies-par-l-utilisateur--udtf-

Chapitre 7: Fonctions définies par l'utilisateur pour la ruche (UDF)

Examples

Création de la ruche UDF

Pour créer un fichier UDF, nous devons étendre la classe UDF (`org.apache.hadoop.hive.ql.exec.UDF`) et implémenter la méthode d'évaluation.

Une fois le fichier UDF respecté et le fichier JAR généré, nous devons ajouter jar au contexte de la ruche pour créer une fonction temporaire / permanente.

```
import org.apache.hadoop.hive.ql.exec.UDF;

class UDFExample extends UDF {

  public String evaluate(String input) {

    return new String("Hello " + input);
  }
}

hive> ADD JAR <JAR NAME>.jar;
hive> CREATE TEMPORARY FUNCTION helloworld as 'package.name.UDFExample';
hive> select helloworld(name) from test;
```

Hive UDF pour couper la chaîne donnée.

```
package MyHiveUDFs;

import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public class Strip extends UDF {

private Text result = new Text();
 public Text evaluate(Text str) {
 if(str == null) {
 return null;
 }
 result.set(StringUtils.strip(str.toString()));
 return result;
 }
}
```

exporter le fichier ci-dessus vers jar

Accédez à l'interface de ligne de commande Hive et ajoutez le fichier JAR UDF

```
hive> ADD jar /home/cloudera/Hive/hive_udf_trim.jar;
```

Vérifiez que JAR se trouve dans Hife CLI Classpath

```
hive> list jars;
/home/cloudera/Hive/hive_udf_trim.jar
```

Créer une fonction temporaire

```
hive> CREATE TEMPORARY FUNCTION STRIP AS 'MyHiveUDFs.Strip';
```

Sortie UDF

```
 hive> select strip('   hiveUDF ') from dummy;
 OK
 hiveUDF
```

Lire Fonctions définies par l'utilisateur pour la ruche (UDF) en ligne:
https://riptutorial.com/fr/hive/topic/4949/fonctions-definies-par-l-utilisateur-pour-la-ruche--udf-

Chapitre 8: Formats de fichier dans HIVE

Examples

SEQUENCEFILE

Stockez les données dans SEQUENCEFILE si les données doivent être compressées. Vous pouvez importer des fichiers texte compressés avec Gzip ou Bzip2 directement dans une table stockée sous TextFile. La compression sera détectée automatiquement et le fichier sera décompressé à la volée pendant l'exécution de la requête.

```
CREATE TABLE raw_sequence (line STRING)
STORED AS SEQUENCEFILE;
```

ORC

Le format de fichier ORC (Optimized Row Columnar) offre un moyen très efficace de stocker des données Hive. Il a été conçu pour surmonter les limitations des autres formats de fichiers Hive. L'utilisation de fichiers ORC améliore les performances lorsque Hive lit, écrit et traite des données. Le fichier ORC peut contenir des index légers et des filtres de floraison.

Voir: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC

ORC est un format recommandé pour stocker des données dans la distribution HortonWorks.

```
CREATE TABLE tab_orc (col1 STRING,
                      col2 STRING,
                      col3 STRING)
STORED AS ORC
TBLPROPERTIES (
               "orc.compress"="SNAPPY",
               "orc.bloom.filter.columns"="col1",
               "orc.create.index" = "true"
              )
```

Pour modifier une table afin que les nouvelles partitions de la table soient stockées en tant que fichiers ORC:

```
ALTER TABLE T SET FILEFORMAT ORC;
```

A partir de Hive 0.14, les utilisateurs peuvent demander une fusion efficace de petits fichiers ORC en émettant une commande `CONCATENATE` sur leur table ou leur partition. Les fichiers seront fusionnés au niveau de la bande sans reserializatoin.

```
ALTER TABLE T [PARTITION partition_spec] CONCATENATE;
```

PARQUET

Format de stockage en colonnes de parquet dans Hive 0.13.0 et versions ultérieures. Le parquet est conçu à partir des structures de données imbriquées complexes et utilise l'algorithme de destruction et d'assemblage d'enregistrements décrit dans l'article de Dremel. Nous pensons que cette approche est supérieure à la simple mise à plat des espaces de noms imbriqués.

Le parquet est conçu pour prendre en charge des schémas de compression et d'encodage très efficaces. Plusieurs projets ont démontré l'impact sur les performances de l'application du bon schéma de compression et d'encodage aux données. Parquet permet de spécifier des schémas de compression au niveau de chaque colonne et est pérenne pour permettre d'ajouter plus d'encodages au fur et à mesure de leur invention et de leur implémentation.

Le parquet est recommandé Format de fichier avec des tables Impala dans les distributions Cloudera.

Voir: http://parquet.apache.org/documentation/latest/

```
CREATE TABLE parquet_table_name (x INT, y STRING) STORED AS PARQUET;
```

AVRO

Les fichiers Avro sont pris en charge dans Hive 0.14.0 et versions ultérieures.

Avro est une structure d'appel de procédure distante et de sérialisation de données développée dans le projet Hadoop d'Apache. Il utilise JSON pour définir les types de données et les protocoles et sérialise les données dans un format binaire compact. Son utilisation principale se situe dans Apache Hadoop, où il peut fournir à la fois un format de sérialisation pour les données persistantes et un format filaire pour la communication entre les nœuds Hadoop et entre les programmes clients et les services Hadoop.

Spécification du format AVRO: https://avro.apache.org/docs/1.7.7/spec.html

```
CREATE TABLE kst
PARTITIONED BY (ds string)
STORED AS AVRO
TBLPROPERTIES (
  'avro.schema.url'='http://schema_provider/kst.avsc');
```

Nous pouvons également utiliser la syntaxe ci-dessous sans utiliser de fichier de schéma.

```
CREATE TABLE kst (field1 string, field2 int)
PARTITIONED BY (ds string)
STORED AS AVRO;
```

Dans les exemples ci-dessus, la clause `STORED AS AVRO` équivaut à:

```
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT
```

```
  'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
```

Fichier texte

TextFile est le format de fichier par défaut, sauf si le paramètre de configuration hive.default.fileformat a un paramètre différent. Nous pouvons créer une table sur ruche en utilisant les noms de champs dans notre fichier texte délimité. Disons par exemple, notre fichier csv contient trois champs (id, nom, salaire) et nous voulons créer une table en ruche appelée "employés". Nous allons utiliser le code ci-dessous pour créer la table dans la ruche.

```
CREATE TABLE employees (id int, name string, salary double) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',';
```

Maintenant, nous pouvons charger un fichier texte dans notre table:

```
LOAD DATA LOCAL INPATH '/home/ourcsvfile.csv' OVERWRITE INTO TABLE employees;
```

Afficher le contenu de notre table sur la ruche pour vérifier si les données ont été chargées avec succès:

```
SELECT * FROM employees;
```

Lire Formats de fichier dans HIVE en ligne: https://riptutorial.com/fr/hive/topic/4513/formats-de-fichier-dans-hive

Chapitre 9: Indexage

Examples

Structure

```
CREATE INDEX index_name
ON TABLE base_table_name (col_name, ...)
AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property_name=property_value, ...)]
[IN TABLE index_table_name]
[PARTITIONED BY (col_name, ...)]
[
 [ ROW FORMAT ...] STORED AS ...
 | STORED BY ...
]
[LOCATION hdfs_path]
[TBLPROPERTIES (...)]
```

Exemple:

```
CREATE INDEX inedx_salary ON TABLE employee(salary) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;
```

Alter Index

ALTER INDEX nom_index ON nom_table [PARTITION (...)] REBUILD

Drop Index

```
DROP INDEX <index_name> ON <table_name>
```

Si WITH DEFERRED REBUILD est spécifié dans CREATE INDEX, l'index nouvellement créé est initialement vide (que la table contienne ou non des données).

La commande ALTER INDEX REBUILD peut être utilisée pour créer la structure d'index pour toutes les partitions ou une seule partition.

Lire Indexage en ligne: https://riptutorial.com/fr/hive/topic/6365/indexage

Chapitre 10: Insérer un relevé

Syntaxe

- **Syntaxe standard:**
- INSERT OVERWRITE TABLE nomtable1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [IF NOT EXISTS]] select_statement1 FROM from_statement;
- INSERT INTO TABLE nomtable1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] select_statement1 FROM from_statement;
- INSERT INTO TABLE nomtable1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] (z, y) select_statement1 FROM from_statement;
- **Extension de ruche (insertions multiples):**
- FROM from_statement
  INSERT OVERWRITE TABLE nomtable1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [SI PAS EXISTE]] select_statement1
  [INSERT OVERWRITE TABLE nomtable2 [PARTITION ... [SI N'EXISTE PAS]] select_statement2]
  [INSERT INTO TABLE nomtable2 [PARTITION ...] select_statement2] ...;
- FROM from_statement
  INSERT INTO TABLE nomtable1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] select_statement1
  [INSERT INTO TABLE nomtable2 [PARTITION ...] select_statement2]
  [INSERT OVERWRITE TABLE nomtable2 [PARTITION ... [SI N'EXISTE PAS]] select_statement2] ...;
- **Extension Hive (insertions de partition dynamiques):**
- INSERT OVERWRITE TABLE nom_table PARTITION (partcol1 [= val1], partcol2 [= val2] ...) select_statement FROM from_statement;
- INSERT INTO TABLE nom_table PARTITION (partcol1 [= val1], partcol2 [= val2] ...) select_statement FROM from_statement;

Remarques

insérer écraser
Une instruction d'insertion d'insertion supprime tous les fichiers existants dans la table ou la partition cible avant d'ajouter de nouveaux fichiers basés sur l'instruction select utilisée. Notez que lorsque des modifications de structure sont apportées à une table ou au DML utilisé pour charger la table, les anciens fichiers ne sont parfois pas supprimés. Lors du chargement dans une table à

l'aide du partitionnement dynamique, seules les partitions définies par l'instruction select seront remplacées. Toutes les partitions préexistantes dans la cible resteront et ne seront pas supprimées.

insérer dans
Une instruction insert in ajoute de nouvelles données dans une table cible en fonction de l'instruction select utilisée.

Examples

insérer écraser

```
insert overwrite table yourTargetTable select * from yourSourceTable;
```

Insérer dans le tableau

INSERT INTO va ajouter à la table ou à la partition, en conservant les données existantes intactes.

```
    INSERT INTO table yourTargetTable SELECT * FROM yourSourceTable;
```

Si une table est partitionnée, nous pouvons insérer dans cette partition particulière de manière statique, comme indiqué ci-dessous.

```
    INSERT INTO TABLE yourTargetTable PARTITION (state=CA, city=LIVERMORE)
    select * FROM yourSourceTable;
```

Si une table est partitionnée, nous pouvons insérer dans cette partition particulière de manière dynamique, comme indiqué ci-dessous. Pour effectuer des insertions de partition dynamique, vous devez définir les propriétés ci-dessous.

```
    Dynamic Partition inserts are disabled by default. These are the relevant configuration
properties for dynamic partition inserts:
    SET hive.exec.dynamic.partition=true;
    SET hive.exec.dynamic.partition.mode=non-strict

    INSERT INTO TABLE yourTargetTable PARTITION (state=CA, city=LIVERMORE) (date,time)
    select * FROM yourSourceTable;
```

Insertion multiple dans une table.

Extension de ruche (insertions multiples):

```
    FROM table_name

    INSERT OVERWRITE TABLE table_one SELECT table_name.column_one,table_name.column_two

    INSERT OVERWRITE TABLE table_two SELECT table_name.column_two WHERE table_name.column_one
== 'something'
```

Lire Insérer un relevé en ligne: https://riptutorial.com/fr/hive/topic/1744/inserer-un-releve

Chapitre 11: Instruction SELECT

Syntaxe

- SELECT [TOUS | DISTINCT] select_expr, select_expr, select_expr,….
- FROM table_reference
- [WHERE où_condition]
- [GROUP BY col_list]
- [Ayant la condition]
- [ORDER BY col_list]
- [LIMIT n]

Examples

Sélectionner toutes les lignes

`SELECT` est utilisé pour récupérer des lignes de données à partir d'une table. Vous pouvez spécifier les colonnes à récupérer:

```
SELECT Name, Position
FROM Employees;
```

Ou utilisez simplement * pour obtenir toutes les colonnes:

```
SELECT *
FROM Employees;
```

Sélectionner des lignes spécifiques

Cette requête renvoie toutes les colonnes des `sales` de la table où les valeurs du `amount` la colonne sont supérieures à 10 et celles de la colonne de la `region` de "US".

```
SELECT * FROM sales WHERE amount > 10 AND region = "US"
```

Vous pouvez utiliser *des expressions régulières* pour sélectionner les colonnes que vous souhaitez obtenir. L'instruction suivante récupérera les données du `name` de la colonne et toutes les colonnes commençant par l' `address` du préfixe.

```
SELECT name, address.* FROM Employees
```

Vous pouvez également utiliser le mot-clé `LIKE` (associé au caractère '%') pour faire correspondre les chaînes qui commencent ou se terminent par une sous-chaîne particulière. La requête suivante renverra toutes les lignes où la `city` colonne commence par "Nouveau"

```
SELECT name, city FROM Employees WHERE city LIKE 'New%'
```

Vous pouvez utiliser le mot-clé `RLIKE` pour utiliser *des expressions régulières* Java. La requête suivante renverra des lignes dont le `name` colonne contient les mots "smith" ou "son".

```
SELECT name, address FROM Employee WHERE name RLIKE '.*(smith|son).*'
```

Vous pouvez appliquer des fonctions aux données renvoyées. La phrase suivante renverra tout le nom en majuscule.

```
SELECT upper(name) FROM Employees
```

Vous pouvez utiliser différentes fonctions mathématiques, fonctions de collecte, fonctions de conversion de type , fonctions de date, fonctions conditionnelles ou fonctions de chaîne .

Afin de limiter le nombre de lignes données dans le résultat, vous pouvez utiliser le mot clé `LIMIT` . L'instruction suivante ne renverra que dix lignes.

```
SELECT * FROM Employees LIMIT 10
```

Sélectionnez: Projeter les colonnes sélectionnées

Structure de la table d'échantillons (par exemple employé)

Nom de colonne	Type de données
ID	INT
F_Name	CHAÎNE
L_Name	CHAÎNE
Téléphone	CHAÎNE
Adresse	CHAÎNE

Projeter toutes les colonnes

Utilisez un joker `*` pour projeter toutes les colonnes. par exemple

```
Select * from Employee
```

Projeter les colonnes sélectionnées (dire ID, nom)

Utilisez le nom des colonnes dans la liste de projection. par exemple

```
Select ID, Name from Employee
```

Jeter 1 colonne de la liste de projection

Afficher toutes les colonnes sauf 1 colonne. par exemple

```
Select `(ID)?+.+` from Employee
```

Supprimer le motif correspondant aux colonnes

Rejeter toutes les colonnes qui correspondent au modèle. p. ex. Rejeter toutes les colonnes se terminant par `NAME`

```
Select `(.*NAME$)?+.+` from Employee
```

Lire Instruction SELECT en ligne: https://riptutorial.com/fr/hive/topic/4133/instruction-select

Chapitre 12: Script de création de table avec des exemples de données

Examples

Types de date et d'horodatage

```
CREATE TABLE all_datetime_types(
  c_date date,
  c_timestamp timestamp
);
```

Valeurs minimales et maximales des données:

```
insert into all_datetime_types values ('0001-01-01','0001-01-01 00:00:00.000000001');
insert into all_datetime_types values ('9999-12-31','9999-12-31 23:59:59.999999999');
```

Types de texte

```
CREATE TABLE all_text_types(
  c_char char(255),
  c_varchar varchar(65535),
  c_string string
);
```

Données d'échantillon:

```
insert into all_text_type values ('some ****&&&%%% char value ','some $$$$#####@@@@ varchar
value','some !!~~~++ string value' );
```

Types numériques

```
CREATE TABLE all_numeric_types(
  c_tinyint tinyint,
  c_smallint smallint,
  c_int int,
  c_bigint bigint,
  c_decimal decimal(38,3)
);
```

Valeurs minimales et maximales des données:

```
insert into all_numeric_types values (-128,-32768,-2147483648,-9223372036854775808,-
99999999999999999999999999999999999.999);
insert into all_numeric_types values
(127,32767,2147483647,9223372036854775807,99999999999999999999999999999999999.999);
```

Types numériques à virgule flottante

```
CREATE TABLE all_floating_numeric_types(
  c_float float,
  c_double double
);
```

Valeurs minimales et maximales des données:

```
insert into all_floating_numeric_types values (-3.4028235E38,-1.7976931348623157E308);
insert into all_floating_numeric_types values (-1.4E-45,-4.9E-324);
insert into all_floating_numeric_types values (1.4E-45,4.9E-324);
insert into all_floating_numeric_types values (3.4028235E38,1.7976931348623157E308);
```

Types booléens et binaires

```
CREATE TABLE all_binary_types(
  c_boolean boolean,
  c_binary binary
);
```

Données d'échantillon:

```
insert into all_binary_types values (0,1234);
insert into all_binary_types values (1,4321);
```

Remarque:

- Pour booléen, en interne, il est enregistré comme vrai ou faux.
- Pour le binaire, il stockera la valeur codée en base64.

Types complexes

Tableau

```
CREATE TABLE array_data_type(
  c_array array<string>)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY '&';
```

Créer `data.csv` avec des données:

```
arr1&arr2
arr2&arr4
```

Placez `data.csv` dans `/tmp` folder et chargez ces données dans Hive

```
LOAD DATA LOCAL INPATH '/tmp/data.csv' INTO TABLE array_data_type;
```

Ou vous pouvez mettre ce fichier CSV en HDFS par exemple à `/tmp` . Charger des données à partir de CSV à HDFS en utilisant

```
LOAD DATA INPATH '/tmp/data.csv' INTO TABLE array_data_type;
```

CARTE

```
CREATE TABLE map_data_type(
  c_map map<int,string>)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY '&'
  MAP KEYS TERMINATED BY '#';
```

fichier `data.csv` :

```
101#map1&102#map2
103#map3&104#map4
```

Charger les données dans la ruche:

```
LOAD DATA LOCAL INPATH '/tmp/data.csv' INTO TABLE map_data_type;
```

STRUCT

```
CREATE TABLE struct_data_type(
  c_struct struct<c1:smallint,c2:varchar(30)>)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY '&';
```

fichier `data.csv` :

```
101&struct1
102&struct2
```

Charger les données dans la ruche:

```
LOAD DATA LOCAL INPATH '/tmp/data.csv' INTO TABLE struct_data_type;
```

UNIONTYPE

```
CREATE TABLE uniontype_data_type(
  c_uniontype uniontype<int, double, array<string>)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY '&';
```

fichier `data.csv` :

```
0&10
1&10.23
2&arr1&arr2
```

Charger les données dans la ruche:

```
LOAD DATA LOCAL INPATH '/tmp/data.csv' INTO TABLE uniontype_data_type;
```

Lire Script de création de table avec des exemples de données en ligne: https://riptutorial.com/fr/hive/topic/5067/script-de-creation-de-table-avec-des-exemples-de-donnees

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec la ruche	Bhavesh, Community, franklinsijo, johnnyaug, NeoWelkin
2	Création de table de ruche à travers Sqoop	NeoWelkin
3	Créer une déclaration de base de données et de table	CodingInCircles, dev ツ, goks, Panther, Venkata Karthik
4	Exporter des données dans la ruche	Prem Singh Bist
5	Fonctions d'agrégat définies par l'utilisateur (UDAF)	dev ツ, Venkata Karthik
6	Fonctions de table définies par l'utilisateur (UDTF)	Venkata Karthik
7	Fonctions définies par l'utilisateur pour la ruche (UDF)	Ashok, Panther, Venkata Karthik
8	Formats de fichier dans HIVE	agentv, Alex, Community, johnnyaug, leftjoin, Mzzzzzz, NeoWelkin, tomek, Venkata Karthik
9	Indexage	Prem Singh Bist
10	Insérer un relevé	Jared, johnnyaug, Venkata Karthik
11	Instruction SELECT	Ambrish, dev, Jaime Caffarel, johnnyaug
12	Script de création de table avec des exemples de données	dev ツ