Interrogation des données avec Pig et Hive

Hmida HMIDA

18 octobre 2022

Motivation

- Programmes MapReduce
  - Avec Java
    - Nombre de lignes de code
  - Avec Python
    - Streaming : difficulté pour mettre en place des programmes complexes
- Background SQL
  - Standard répandu
  - Exploiter les ressources existantes
    - Code
    - Ingénieurs/développeurs

1 Pig

Présentation

- Apache Pig est un logiciel initialement créé par Yahoo!. En 2006
  - Il permet d'écrire des traitement utiles sur des données, sans subir la complexité de Java.
  - Le but est de rendre Hadoop accessible à des non-informaticiens scientifiques : physiciens, statisticiens, mathématiciens. . .
  - Projet Apache en 2007 : http://pig.apache.org
- Pig propose un langage de scripts appelé « Pig Latin ».
  - Ce langage est qualifié de « Data Flow Language ».
- Pig traduit les scripts Pig Latin en jobs MapReduce
- Peut être intégré à d'autres langages

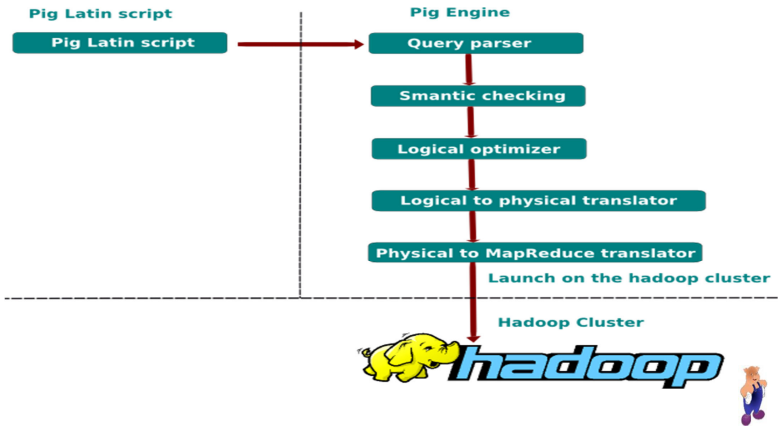
5

Présentation (2)

- Pig utilise des données provenant de nombreuses sources, y compris des données structurées et non structurées.
- Pig stocke les résultats dans le système de fichiers HDFS de Hadoop.
- Pig est composé de 2 modules qui sont :
  - Pig Latin : langage procédural de haut niveau.
  - Pig Engine : parse, optimise et exécute automatiquement les scripts Pig Latin comme une série de jobs MapReduce au sein d'un cluster Hadoop.

6

Architecture Pig



7

Exécution de scripts Pig

- 2 modes
  - Mode local : pig –x local
    - mode de test sur la machine locale
    - Pas de Hadoop ou HDFS
  - Mode MapReduce : pig –x mapreduce
    - Jobs MapReduce en arrière-plan
    - Opération sur les données qui existent dans le HDFS.
- 3 outils
  - Grub : une interface CLI interactive
  - Exécuter un script .pig
  - API Java

8

Exécution de scripts Pig (2)

- Exécution de fichier script
  - Avec un éditeur de textes :

```
/* idusers.pig */
A = load 'passwd' using PigStorage(':'); -- load the passwd file
B = foreach A generate $0 as id; -- extract the user IDs
store B into 'id.out'; -- write the results to a file name id.out
```

  - Exécuter le script

```
pig idusers.pig
```

- Shell interactif (PigLatin et commandes HDFS)

```
pig
…connecting to…
grunt> lines = LOAD './shakespeare.txt' AS (line: chararray);
grunt> DESCRIBE lines;
lines: {line: chararray}
```

Exécution de scripts Pig (3)

- API Java
  - Ajouter pig.jar
  - Exemple :

```
import java.io.IOException;
import org.apache.pig.PigServer ;
public class WordCount {
    public static void main(String[] args ){
        PigServer pigServer = new PigServer ();
        try{
            pigServer.registerJar("mylocation /tokenize.jar");
            runMyQuery (pigServer , "myinput.txt";
        }
        catch (IOException e) { e.printStackTrace (); }
        public static void runMyQuery (PigServer pigServer , String inputFile ) throws IOException {
            pigServer.registerQuery("A = load '" + inputFile + "' using TextLoader ();");
            pigServer.registerQuery("B = foreach A generate flatten tokenize ($0));");
            pigServer.registerQuery("C = group B by $1;");
            pigServer.registerQuery("D = foreach C generate flatten (group), COUNT(B.$0);");
            pigServer.store("D", myoutput ");
        }
    }
}
```


Modèle de données Pig

- Types de données scalaires

Type simple	Description
Int	Il représente les entiers. Il est représenté par l'interface java.lang.Integer. Il est codé sur sur 4 bits signées.
Long	il représente les entiers de type long. Il est représenté par l'interface java.lang.Long. Il est codé sur 8 bits signés.
Float	il désigne les données de type réel. Il est représenté par l'interface java.lang.Float. Il est codé sur 5 bits.
Double	il désigne les données de type réels (double précision). Il est représenté par l'interface java.lang.Double. Il est codé sur 8 bits.
Chararray	ce dernier permet de représenter les string ou un tableau de caractère. Il est représenté par l'interface java.lang.String. Le chaines constantes peuvent etre exprimés par un simple quotte. Les caractères de contrôles doivent etre précédés par « \ ».
Bytearray	il représente une un tableau de byte. Il est représenté par une classe java, appelé DataByteArray, qui a son tour encapsule un tableau de bits (byte[]).

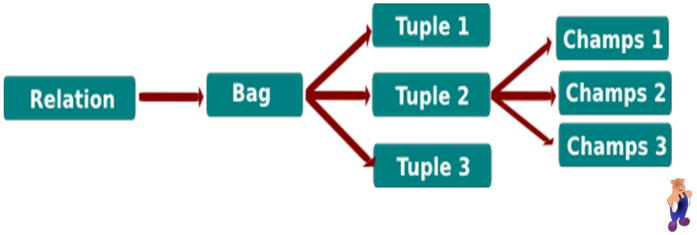
Modèle de données Pig (2)

- Types de données complexes

Type complexe	Description
Map *[key1#val1,…..key2#val2]*	Il s'agit d'un map, composé d'une clé et une valeur.
Tuple *(field1,…..,fieldn)*	Il s'agit d'une collection ordonnée.
Bag *{tuple1, … , tuplen}*	Il s'agit d'une collection de tuple non ordonnée, c-a-d, la notion de position n'est pas supportée.

Modèle de données Pig (3)

- Relation
  - Une relation est un bag de tuples.
  - Les relations en Pig n'exigent pas que chaque tuple doit contenir le même nombre de champs ou les champs situés dans la même position (colonne) doivent avoir le même type.



13

Modèle de données Pig (4)

- Schéma de relation
  - La liste des champs d'une relation est appelé schéma. C'est un n-uplet. On l'écrit (nom1:type1, nom2:type2, ...)
  - Par exemple, une relation contenant des employés aura le schéma suivant :

```
(id:long,  nom:chararray,  prenom:chararray,  photo:bytearray, ancienneté:int, salaire:float)
```

- Accès aux champs d'une relation :
  - Soit par leur nom en clair
  - Soit par leur position $0 désignant le premier champ, $1 le deuxième et ainsi de suite.
  - Quand il y a ambiguïté sur la relation : relation.champ
  - Lorsqu'un champ est un tuple, ses éléments sont nommés relation.champ.element.
    - Par exemple segments.P1.z1
    - Pour un champ de type map, ses éléments sont nommés relation.champ#element. Par exemple heros.infos#metier
    - Il n'y a pas de syntaxe pour l'accès aux champs de type bag.

14

Pig Latin (1)

- Les commentaires sont placés entre /*...*/ ou à partir de -- et la fin de ligne.
- Un programme Pig Latin est une succession d'instructions. Toutes doivent être terminées par un ;
- Pas de variables, ni de fonctions/procédures.
- Le résultat de chaque instruction Pig est une relation.
  - sortie = INSTRUCTION entree PARAMETRES ...;
- Un programme :
  - Chargement avec LOAD
  - Succession de transformations de relations
  - Récupérer le résultat avec DUMP ou STORE

15

Pig Latin (2)

- Instructions

Opérateur	Description	Exemple
LOAD	L'opérateur LOAD est utilisé pour charger les données du système de fichiers ou du stockage HDFS dans une relation Pig.	A = LOAD 'student' AS (name:chararray, age:int);
FOREACH	L'opérateur FOREACH génère des transformations de données basées sur des colonnes de données.	B= FOREACH A GENERATE name;
FILTER	Cet opérateur sélectionne des tuples dans une relation basée sur une condition.	C = FILTER A by age>20;
JOIN	L'opérateur JOIN effectue toujours une jointure interne.	D = JOIN A by name , B by name;
ORDER BY	L'opérateur ORDERBY est utilisé pour trier une relation en fonction d'un ou plusieurs champs	E = ORDER A by age;
STORE	L'opérateur Store est utilisé pour enregistrer les résultats dans le système de fichiers.	X = STORE D into 'output'

16

Chargement de fichiers

```
LOAD [LOCAL] 'emplacement' [USING PigStorage(<sep>)] [AS <schéma>];
```

- LOCAL : pour un chemin local sinon c'est sur HDFS
- PigStorage pour le format CSV
- Exemple 1 (tuples)

S1 (3,8,9) (4,5,6)
S2 (1,4,7) (3,7,5)
S3 (2,5,8) (9,5,8)

```
segments = LOAD 'segments.csv' AS (
    nom:chararray,
    P1:tuple(x1:int, y1:int, z1:int),
    P2:tuple(x2:int, y2:int, z2:int)
);
```

17

Chargement de fichiers (2)

- Exemple 2 (bags)

```
L1  {(3,8,9),(4,5,6)}
L2  {(4,8,1),(6,3,7),(7,4,5),(5,2,9),(2,7,1)}
L3  {(4,3,5),(6,7,1),(3,1,7)}
```

```
segments = LOAD 'segments.csv' AS (
    nom:chararray,
    Points:{tuple(x:int,  y:int,  z:int)}
);
```

- Exemple 3 (maps)

```
1   [nom#asterix,metier#guerrier]
2   [nom#tintin,metier#journaliste]
3   [nom#spirou,metier#groom]
```

```
heros = LOAD 'heros.csv' AS (
    id:int,
    infos:map[chararray]
);
```

18

Enregistrement ou affichage

```
STORE relation INTO 'fichier' USING PigStorage('sep');
```

- Enregistre la relation dans le fichier, sous forme d'un fichier CSV séparé par sep, ex: 1;1, 1:1. . .

```
DUMP relation;
```

- Lance le calcul MapReduce de la relation et affiche les résultats à l'écran. C'est seulement à ce stade que la relation est calculée.

```
SAMPLE relation;
```

- Affiche quelques n-uplets choisis au hasard, une sorte d'échantillon de la relation.

```
DESCRIBE relation;
```

- Affiche le schéma, c'est à dire les champs, noms et types, de la relation. C'est à utiliser dès qu'on a un doute sur le programme.

19

ORDER, LIMIT, DISTINCT

```
ORDER relation BY champ [ASC|DESC], ...;
```

- Tri selon un ou plusieurs champs.

```
LIMIT relation N;
```

- Garder les N premiers tuples.

```
DISTINCT relation;
```

- Éliminer les doublons.

- Exemple :
  - Afficher les 3 produits plus chers

```
produits = LOAD 'produits.csv' USING PigStorage(',') AS (id:int, nom:chararray, prix:double, stock:int);
produitstries = ORDER produits BY prix DESC;
top3 = LIMIT produitstries 3;
-- ou par imbrication d'instructions
top3 = LIMIT (ORDER (LOAD 'produits.csv' USING PigStorage(',') AS (id:int, nom:chararray, prix:double, stock:int)) BY prix DESC) 3;
```

20

FILTER

- `FILTER relation BY condition;`
  - La condition :
    - Opérateurs de comparaison : <, <=, >, >=, !=
    - Opérateurs logiques : AND, OR, NOT
    - Vérifier NULL : IS NULL, IS NOT NULL
  - Exemple : les produits dont le prix est dans [100,200]

```
produits100_200 = FILTER produits by prix>=100 AND prix<=200;
```

21

FOREACH GENERATE

```
FOREACH relation GENERATE exp [AS champ], ....;
```

- Insctruction permettant de faire :
  - Projection

```
FOREACH produit GENERATE nom, prix;
-- garder les champs nom et prix
FOREACH produit GENERATE nom ..;
-- garder les champs nom et ceux qui le suivent
```

  - Calculs à partir d'une relation

```
FOREACH relation GENERATE nom, prix*0.8 as prix_promo;
-- calculer le prix après une remise de 20%
```

22

GROUP BY

- `GROUP relation BY champ;`
  - Regroupe les tuples ayant la même valeur pour un champ
  - Crée une relation de couples : 2 champs
    - group : prend la valeur commune du champ
    - relation : le même nom que la relation, un Bag de tuples
- Exemple :

```
commandes = LOAD 'commandes.csv' USING PigStorage(',') AS
(numCmd:int, numClient:int, montant:double);
cmd_client = GROUP commandes BY numClient;
```

```
1, 1, 12.50
2, 2, 21.75
3, 3, 56.25
4, 1, 34.00
5, 3, 3.30
```

commandes.csv

group | commandes

```
(1, {(4,1,34.0), (1,1,12.5)})
(2, {(2,2,21.75)})
(3, {(3,3,56.25), (5,3,3.30)})
```

cmd_client

23

Agrégats

- S'appliquent aux groupes créés par GROUP BY

```
FOREACH relation GENERATE agrégat(bag.champ);
```

- Exemple :
  - Somme des montants des commandes par client

```
FOREACH cmd_client GENERATE SUM(commandes.montant);
```

- Agrégats :
  - SUM, AVG, MIN, MAX, COUNT
- Exemple de requête complexe : Nombre de commandes à montant >=500

```
commandes = LOAD 'commandes.csv' USING PigStorage(',') AS (numCmd:int,
numClient:int, montant:double);
cmd_client = GROUP commandes BY numClient;
nb_big_cmd = FOREACH cmd_client {
big_cmd = FILTER commandes BY montant>=500.0;
GENERATE group, COUNT(big_cmd.numCmd) AS nb_cmd;
};
```

24

Jointures

```
JOIN relation1 BY champ1 [LEFT | RIGHT], relation2 BY champ2;
```

- Exemple :
  - Jointure entre les clients et leurs commandes

```
JOIN clients BY numClient, commandes BY numClient;
```

  - Jointure entre les clients et leurs commandes mêmes les clients sans commandes

```
JOIN clients BY numClient LEFT, commandes BY numClient;
```

- Produit cartésien

```
CROSS relation1, relation2 ...;
```

25

Autres fonctions

- Sur les bags :
  - FLATTEN(bag) : obtenir une liste plate des éléments de bag
  - SIZE(bag) : taille
  - DIFF(bag1, bag2) : éléments non communs
- Chaines de caractères :
  - SPLIT, UPPER, LOWER, SUBSTRING, STARTSWITH, ENDSWITH, ...
- Mathématiques :
  - ABS, FLOOR, CEIL, ...
- Dates :
  - ADDDURATION, DAYSBETWEEN, ..., GETDAY,...
- User Defined Functions : Python, Java, ...

2 Hive

27

Présentation

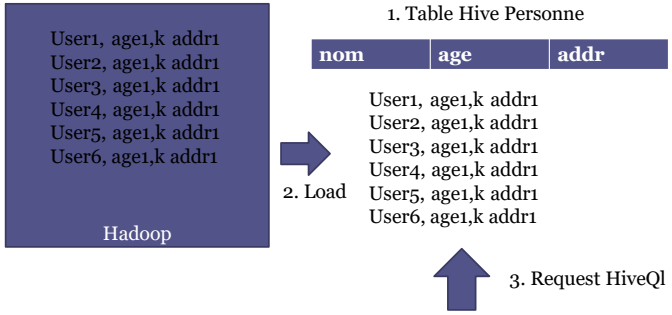
- Hadoop
  - Idéal pour le stockage et le traitement de quantités massives de données.
  - L'extraction de données est complexe et coûteuse.
- Solution : Apache Hive.
- Apache Hive est un logiciel de Data Warehouse initialement créé par Facebook.
- Effectuer des requêtes « SQL-like » pour extraire efficacement des données en provenance de Apache Hadoop.
- Hive n'est pas une base de données ni un datawarehouse classique :
  - Il s'agit d'un système qui maintient des métadonnées décrivant les données stockées dans HDFS.
  - Il utilise une base de données relationnelle appelée metastore (Derby par défaut) pour assurer la persistance des métadonnées.
  - Traduit les requêtes en Jobs MapReduce (ou aussi Tez, Spark).

28

Caractéristiques

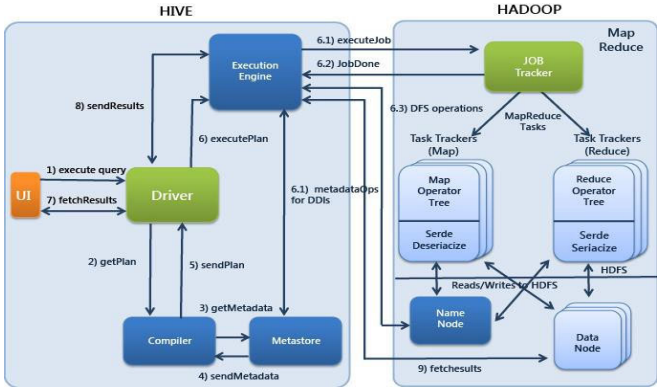
- Hive est un moteur d'entrepôt de données :
  - Tout d'abord, l'utilisateur doit créer les tables, puis charger les données pour soumettre les requêtes au-dessus des tables.
- Hive fait l'optimisation des requêtes qui fait référence à un moyen efficace d'exécution des requêtes en termes de performances comme le formatage de fichiers, le partitionnement , le bucketing (partager les liens du blog).
- HiveQL est un avantage pour l'utilisateur en raison de la complexité de MapReduce.
- Hive est proche du monde de bases de données relationnelles.
- Les requêtes Hive peuvent être exécutées à l'aide de l'interface de ligne de commande hive, des scripts du langage de requête hive (.hql) et à l'aide d'une API java.
- Hive prend en charge différents formats de fichiers comme TEXTFILE, SEQUENCE FILE, ORC et RCFILE (Record Columnar File).

Fonctionnement de Hive



Architecture

- Hive possède des interface pour les langages de programmation :
  - ThriftServer : Remote Procedure Call
  - JDBC



Modèle de données

- Les données dans Hive sont organisées en :
  - Bases de données : namespace de haut niveau pour les tables, vues, ...
  - Tables : Données homogènes avec un schéma
  - Partitions : organisation du stockage des tables selon une clé de partitionnement
  - Buckets ou clusters : 2ème niveau de partitionnement selon un hachage différent de la clé de partitionnement

Schéma d'une table

- Le Langage de Définition de Données
  - Très proche SQL
  - Pas de contraintes d'intégrité
  - Détails sur le fichier source
- Exemple : création de table

```
CREATE TABLE produit (
    idProduit INT,
    nom STRING,
    prix FLOAT,
    qualite INT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

33

Types de données

- Hive définit les types suivants :
  - BIGINT (8 octets), INT (4), SMALLINT (2), TINYINT, (1 octet)
  - FLOAT et DOUBLE
  - BOOLEAN valant TRUE ou FALSE
  - STRING, on peut spécifier le codage (UTF8 ou autre), VARCHAR et CHAR
  - TIMESTAMP exprimé en nombre de secondes.nanosecondes depuis le 01/01/1970 UTC
  - données structurées comme avec Pig :
    - ARRAY<type> indique qu'il y a une liste de type
    - STRUCT<nom1:type1, nom2:type2...> pour une structure regroupant plusieurs valeurs
    - MAP<typecle, typeval> pour une suite de paires clé,valeur

34

Format des fichiers

- La création d'une table se fait ainsi :

```
CREATE TABLE nom (schéma) ROW FORMAT DELIMITED description du format
```

- Les directives situées après le schéma indiquent la manière dont les données sont stockées dans le fichier CSV. Ce sont :
  - FIELDS TERMINATED BY ';' : il y a un ; pour séparer les champs
  - COLLECTION ITEMS TERMINATED BY ',' : il y a un , entre les éléments d'un ARRAY
  - MAP KEYS TERMINATED BY ':' : il y a un : entre les clés et les valeurs d'un MAP
  - LINES TERMINATED BY '\n' : il y a un \n en fin de ligne
  - STORED AS TEXTFILE : c'est un CSV.

35

Chargement de données

- Charger un fichier CSV depuis HDFS dans une table

```
LOAD DATA INPATH '/user/ventes/produits.csv' OVERWRITE INTO TABLE produit;
```

- Hive déplace le fichier source dans ses dossiers
  - Utiliser CREATE EXTERNAL TABLE pour empêcher le délacement
- Charger un fichier local

```
LOAD DATA LOCAL INPATH 'clients.csv' OVERWRITE INTO TABLE client;
```

36

Requêtes HiveQL

- LMD

```
INSERT INTO produit VALUES (10, 'Papier A4', 20.5, 50);
UPDATE produit set prix = prix*0.8;
DELETE FROM produit WHERE id=10;
```

- SELECT

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]
[LIMIT [offset,] rows]
```

Autres requêtes

- Enregistrer le résultats

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/prod/cher' SELECT  idProduit, nom, prix, stock
FROM produit
WHERE prix > 100.0;
```

- Autres commandes
  - DDL
    - CREATE DATABASE/SCHEMA, TABLE, VIEW, FUNCTION, INDEX
    - DROP DATABASE/SCHEMA, TABLE, VIEW, INDEX
    - TRUNCATE TABLE
    - ALTER DATABASE/SCHEMA, TABLE, VIEW
    - SHOW DATABASES/SCHEMAS, TABLES, TBLPROPERTIES, VIEWS, PARTITIONS, FUNCTIONS, INDEX[ES], COLUMNS, CREATE TABLE
    - DESCRIBE DATABASE/SCHEMA, table_name, view_name, materialized_view_name
  - DML
    - LOAD
    - INSERT INTO
    - UPDATE
    - DELETE
    - MERGE
  - EXPLAIN

Hive CLI/beeline

Command	Description
Quit | exit	Use quit or exit to leave the interactive shell.
Reset	Resets the configuration to the default values (as of Hive 0.10: see HIVE-3202).
set <key>=<value>	Sets the value of a particular configuration variable (key). Note: If you misspell the variable name, the CLI will not show an error.
set	Prints a list of configuration variables that are overridden by the user or Hive.
set -v	Prints all Hadoop and Hive configuration variables.
add FILE[S] <filepath> <filepath>* **add JAR[S] <filepath> <filepath>*** **add ARCHIVE[S] <filepath> <filepath>***	Adds one or more files, jars, or archives to the list of resources in the distributed cache.
list FILE[S] [<filepath>*] **list JAR[S] [<filepath>*]** **list ARCHIVE[S] [<filepath>*]**	Lists or Checks whether the given resources are already added to the distributed cache or not.
delete FILE[S] <filepath>* **delete JAR[S] <filepath>*** **delete ARCHIVE[S] <filepath>***	Removes the resource(s) from the distributed cache.
! <command>	Executes a shell command from the Hive shell.
dfs <dfs command>	Executes a dfs command from the Hive shell.
<query string>	Executes a Hive query and prints results to standard output.
source <filepath>	Executes a script file inside the CLI.

Exemple JDBC

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
 public class HiveJdbcClient {
  private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

  public static void main(String[] args) throws SQLException {
    try {
      Class.forName(driverName);
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      System.exit(1);
    }
    Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/default", "", "");
    Statement stmt = con.createStatement();
    String tableName = "testHiveDriverTable";
    stmt.executeQuery("drop table " + tableName);
    ResultSet res = stmt.executeQuery("create table " + tableName + " (key int, value string)");
```


Exemple JDBC (2)

```
    // show tables
    String sql = "show tables '" + tableName + "'";
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    if (res.next()) {
      System.out.println(res.getString(1));
    }
    // describe table
    sql = "describe " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(res.getString(1) + "\t" + res.getString(2));
    }
    // load data into table
    // NOTE: filepath has to be local to the hive server
    // NOTE: /tmp/a.txt is a ctrl-A separated file with two fields per line
    String filepath = "/tmp/a.txt";
    sql = "load data local inpath '" + filepath + "' into table " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
```

Exemple JDBC (3)

```
    // select * query
    sql = "select * from " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(String.valueOf(res.getInt(1)) + "\t" + res.getString(2));
    }

    // regular hive query
    sql = "select count(1) from " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(res.getString(1));
    }
  }
}
```


Exemple Python

```
import sys

from hive import ThriftHive
from hive.ttypes import HiveServerException
from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol

try:
    transport = TSocket.TSocket('localhost', 10000)
    transport = TTransport.TBufferedTransport(transport)
    protocol = TBinaryProtocol.TBinaryProtocol(transport)

    client = ThriftHive.Client(protocol)
    transport.open()

    client.execute("CREATE TABLE r(a STRING, b INT, c DOUBLE)")
    client.execute("LOAD TABLE LOCAL INPATH '/path' INTO TABLE r")
    client.execute("SELECT * FROM r")
    while (1):
      row = client.fetchOne()
      if (row == None):
        break
      print row
    client.execute("SELECT * FROM r")
    print client.fetchAll()

    transport.close()

except Thrift.TException, tx:
    print '%s' % (tx.message)
```