EBook Gratuito

APPRENDIMENTO hive

Free unaffiliated eBook created from
Stack Overflow contributors.

#hive

Sommario

Di

Capitolo 1: Iniziare con l'alveare

Osservazioni

- Hive è uno strumento di data warehouse costruito su Hadoop .
- Fornisce un linguaggio simile a SQL per interrogare i dati.
- Possiamo eseguire quasi tutte le query SQL in Hive, l'unica differenza è che esegue un lavoro di riduzione della mappa sul back-end per recuperare i risultati da Hadoop Cluster. A causa di questo Hive a volte impiega più tempo a recuperare il set di risultati.

Examples

Esempio di conteggio parole in Hive

File di documenti (file di input)

Mary ha un piccolo agnello

il suo vello era bianco come la neve

e dappertutto che Mary è andata

l'agnello era sicuro di andare.

Hive Query

```
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;
```

Output della tabella word_counts in Hive

Mary, 2

aveva, 1

a, 1

poco, 1

agnello, 2

sua, 1

vello, 1

era, 2

bianco, 1

come, 1

neve, 1

e, 1

ovunque, 1

che, 1

è andato, 1

la, 1

certo, 1

a, 1

andare, 1

Installazione di Hive (linux)

Inizia scaricando l'ultima versione stabile da https://hive.apache.org/downloads.html

-> Ora decomprimere il file con

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

-> Crea una directory in / usr / local / con

$ sudo mkdir / usr / local / hive

-> Sposta il file su root con

$ mv ~ / Download / hive-2.xy / usr / local / hive

-> Modifica le variabili d'ambiente per hadoop e hive in .bashrc

$ gedit ~ / .bashrc

come questo

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/* :.

-> Ora, avvia hadoop se non è già in esecuzione. Assicurati che sia in esecuzione e che non sia in modalità provvisoria.

$ hadoop fs -mkdir / user / hive / warehouse

La directory "warehouse" è la posizione in cui archiviare la tabella oi dati relativi all'alveare.

$ hadoop fs -mkdir / tmp

La directory temporanea "tmp" è la posizione temporanea in cui archiviare il risultato intermedio dell'elaborazione.

-> Imposta i permessi per leggere / scrivere su quelle cartelle.

$ hadoop fs -chmod g + w / user / hive / warehouse

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

-> Ora attiva HIVE con questo comando in console

$ hive

Installazione di Hive con Metastore esterno in Linux

Pre-requisiti:

1. Java 7
2. Hadoop (fare riferimento qui per l'installazione di Hadoop)
3. Mysql Server e Client

Installazione:

Passo 1: Scarica l'ultimo tarball di Hive dalla pagina dei download .

Passo 2: Estrai il tarball scaricato (***Presupposto:*** *il tarball viene scaricato in $ HOME*)

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

Passaggio 3: aggiornare il file di ambiente (`~/.bashrc`)

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

fonte il file per impostare le nuove variabili di ambiente.

```
source ~/.bashrc
```

Passo 4: Scarica il connettore JDBC per mysql da qui ed estrai.

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

La directory estratta contiene il file jar del connettore `mysql-connector-java-abcjar` . `$HIVE_HOME` sulla `lib` di `$HIVE_HOME`

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

Configurazione:

Creare il file di configurazione hive `hive-site.xml` nella `$HIVE_HOME/conf/` e aggiungere le seguenti proprietà relative a metastore.

```
<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>
```

Aggiorna i valori di MySQL "username" e "password" di conseguenza nelle proprietà.

Creare lo schema Metastore:

Gli script dello schema metastore sono disponibili in `$HIVE_HOME/scripts/metastore/upgrade/mysql/`

Accedi a MySQL e avvia lo schema,

```
mysql -u username -ppassword

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

Starting Metastore:

```
hive --service metastore
```

Per eseguirlo in background,

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

Avvio di HiveServer2: (utilizzare se richiesto)

```
hiveserver2
```

Per eseguirlo in background,

```
nohup hiveserver2 metastore &
```

Nota: questi file eseguibili sono disponibili in `$HIVE_HOME/bin/`

Collegare:

Usa entrambi `hive` , `beeline` o Hue per connetterti con Hive.

La CLI di Hive è obsoleta, si consiglia di utilizzare Beeline o Hue.

Configurazioni aggiuntive per la tonalità:

Aggiorna questo valore in `$HUE_HOME/desktop/conf/hue.ini`

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

Leggi Iniziare con l'alveare online: https://riptutorial.com/it/hive/topic/1099/iniziare-con-l-alveare

Capitolo 2: Crea un database e una tabella

Sintassi

- CREATE [TEMPORARY] [EXTERNAL] TABLE [SE NON ESISTE] [nome_db.] Nome_tabella

  [(nome_colazione tipo_dati [COMMENTO col_commento], ...)]
  [COMMENTO_composito_commerciale] [PARTITIONED BY (nome_coli_dati tipo
  [COMMENTO col_commento], ...)] [CLUSTERED BY (nome_col, nome_colonna, ...)
  [ORDINATO DA (nome_col [ASC | DESC], ...)] INTO num_buckets BUCKETS] [SKEWED
  BY (col_name, col_name, ...) - (Nota: disponibile in Hive 0.10.0 e versioni successive)] ON
  ((valore_casuale, valore_colore, ...), (col_value, col_value, ...), ...) [STORED AS
  DIRECTORIES] [[ROW FORMAT row_format] [STORED AS file_format] | MEMORIZZATO
  DA "storage.handler.class.name" [WITH SERDEPROPERTIES (...)]] [LOCATION hdfs_path]
  [TBLPROPERTIES (property_name = property_value, ...)]
  [AS select_statement];

- CREATE [TEMPORARY] [EXTERNAL] TABLE [SE NON ESISTE] [nome_db.] Nome_tabella LIKE nome_tabella_esistente_ESTAZIONE [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 (precisione, scala), DATE, VARCHAR, CHAR
- array_type: ARRAY <data_type>
- map_type: MAP <primitive_type, data_type>
- struct_type: STRUCT <col_name: data_type [COMMENT col_comment], ...>
- union_type: UNIONTYPE <data_type, data_type, ...>
- row_format: DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]] [ARTICOLI DELLA COLLEZIONE TERMINATI DA char] [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char] [NULL DEFINED AS char]
  , SERDE serde_name [WITH SERDEPROPERTIES (nome_proprietà = valore_proprietà, nome_proprietà = valore_proprietà, ...)]
- file_formato: SEQUENCEFILE, TEXTFILE, RCFILE, ORC, PARQUET, AVRO, INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname
- CREATE (DATABASE | SCHEMA) [SE NON ESISTE] database_name [COMMENT database_comment] [LOCATION hdfs_path] [WITH DBPROPERTIES (property_name = property_value, ...)];

Osservazioni

Quando si lavora con tabelle e database in HIVE. Sotto i punti può essere utile.

- Possiamo cambiare il database usando il `use database;` comando
- Per conoscere il database di lavoro corrente possiamo usare `SELECT current_database()`
- Per visualizzare il DDL utilizzato per creare la dichiarazione della tabella, è possibile utilizzare `SHOW CREATE TABLE tablename`
- Per vedere tutte le colonne della tabella usa `DESCRIBE tablename` per mostrare dettagli estesi come il serde di posizione usato e altri `DESCRIBE FORMATTED tablename` . DESCRIBE può anche essere abbreviato come DESC.

Examples

Crea tabella

Creazione di una tabella **gestita** con partizione e memorizzata come un file di sequenza. Si presume che il formato dei dati nei file sia delimitato da `Ctrl-A (^A)` e delimitato da newline. La tabella seguente viene creata nella directory warehouse hive specificata nel valore per la chiave `hive.metastore.warehouse.dir` nel file di configurazione 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;
```

Creazione di una tabella **esterna** con partizioni e memorizzata come un file di sequenza. Si presuppone che il formato dei dati nei file sia delimitato dal campo da `ctrl-A` e dalla riga delimitata da newline. La tabella seguente viene creata nella posizione specificata ed è utile quando disponiamo già di dati. Uno dei vantaggi dell'utilizzo di una tabella esterna è che possiamo eliminare la tabella senza eliminare i dati. Ad esempio, se creiamo una tabella e ci rendiamo conto che lo schema è sbagliato, possiamo tranquillamente abbandonare la tabella e ricrearla con il nuovo schema senza preoccuparci dei dati. Un altro vantaggio è che se usiamo altri strumenti come maiale sugli stessi file, possiamo continuare a usarli anche dopo aver eliminato la tabella.

```
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>';
```

Creando una tabella usando select query e popolando i risultati da query, queste istruzioni sono note come **CTAS (Create Table As Select)** .

Ci sono due parti in CTAS, la parte SELECT può essere qualsiasi istruzione SELECT supportata da HiveQL. La parte CREATE di CTAS acquisisce lo schema risultante dalla parte SELECT e crea la tabella di destinazione con altre proprietà della tabella come SerDe e il formato di archiviazione.

Il CTAS ha queste restrizioni:

- La tabella di destinazione non può essere una tabella partizionata.
- La tabella di destinazione non può essere una tabella esterna.
- La tabella di destinazione non può essere una tabella di bucket di elenco.

```
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;
```

Crea una tabella simile a:

La **forma LIKE di CREATE TABLE** consente di copiare esattamente una definizione di tabella esistente (senza copiarne i dati). A differenza di CTAS, la seguente istruzione crea una nuova tabella la cui definizione corrisponde esattamente alla tabella esistente in tutti i particolari diversi dal nome della tabella. La nuova tabella non contiene righe.

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

Crea Database

Creazione di un database in una posizione particolare. Se non specifichiamo alcuna posizione per il database, è stata creata nella directory warehouse.

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

Creazione di tabelle ACID Hive.

Le tabelle ACID sono supportate dalla versione hive 0.14. La tabella sottostante supporta UPDATE / DELETE / INSERT

Sotto le modifiche alla configurazione richieste in 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
```

Attualmente solo il file orc è supportato dal formato.

Tabella creare una dichiarazione.

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

Integrazione HIVE_HBASE

L'integrazione Hive-Hbase è supportata poiché le versioni successive. 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");
```

Nota: la prima colonna dovrebbe essere la colonna chiave.

Crea una tabella usando le proprietà della tabella esistenti.

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

Leggi Crea un database e una tabella online: https://riptutorial.com/it/hive/topic/3328/crea-un-database-e-una-tabella

Capitolo 3: Creazione di tabelle alveare tramite Sqoop

introduzione

Se disponiamo di un meta-store Hive associato al nostro cluster HDFS, Sqoop può importare i dati in Hive generando ed eseguendo un'istruzione CREATE TABLE per definire il layout dei dati in Hive. L'importazione di dati in Hive è semplice come aggiungere l'opzione --hive-import alla riga di comando Sqoop.

Osservazioni

L'importazione di dati direttamente da RDBMS a HIVE può risolvere un sacco di tempo. Inoltre possiamo eseguire una query a mano libera (un join o qualche semplice query) e popolarla in una tabella di nostra scelta direttamente in Hive.

> --hive-import dice a Sqoop che la destinazione finale è Hive e non HDFS.
>
> L'opzione --hive-table aiuta a importare i dati nella tabella in hive scelti da noi, altrimenti verrà chiamato come tabella di origine importata da RDBMS.

Examples

Importazione dell'alveare con il nome della tabella di destinazione nell'alveare

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

Leggi Creazione di tabelle alveare tramite Sqoop online:
https://riptutorial.com/it/hive/topic/10685/creazione-di-tabelle-alveare-tramite-sqoop

Capitolo 4: Esportare i dati in Hive

Examples

Esporta funzione nell'alveare

Esportazione dei dati dalla tabella dei dipendenti in / tmp / ca_dipendenti

INSERISCI SOVRASTRO LOCAL DIRECTORY '/ tmp / ca_employees' SELECT nome, salario, indirizzo Dipendenti WHERE se.state = 'CA';

Esportazione di dati dalla tabella dei dipendenti in più directory locali in base a condizioni specifiche

La query seguente mostra come un singolo costrutto può essere utilizzato per esportare i dati in più directory in base a criteri specifici

```
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';
```

Leggi Esportare i dati in Hive online: https://riptutorial.com/it/hive/topic/6530/esportare-i-dati-in-hive

Capitolo 5: Formati di file in HIVE

Examples

SEQUENCEFILE

Memorizza i dati in SEQUENCEFILE se i dati devono essere compressi. Puoi importare file di testo compressi con Gzip o Bzip2 direttamente in una tabella memorizzata come TextFile. La compressione verrà rilevata automaticamente e il file verrà decompresso al volo durante l'esecuzione della query.

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

ORC

Il formato di file Ottimized Row Columnar (ORC) fornisce un modo estremamente efficiente per archiviare i dati Hive. È stato progettato per superare i limiti degli altri formati di file Hive. L'uso dei file ORC migliora le prestazioni quando Hive legge, scrive e elabora i dati. Il file ORC può contenere indici leggeri e filtri bloom.

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

ORC è un formato consigliato per la memorizzazione dei dati nella distribuzione di 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"
              )
```

Per modificare una tabella in modo che le nuove partizioni della tabella siano memorizzate come file ORC:

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

A partire da Hive 0.14, gli utenti possono richiedere un'efficace unione di piccoli file ORC mediante l'emissione di un comando `CONCATENATE` sulla propria tabella o partizione. I file verranno uniti a livello di strip senza reserializatoin.

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

PARQUET

Formato di archiviazione colonnare del parquet in Hive 0.13.0 e versioni successive. Parquet è costruito da zero con strutture di dati nidificati complessi in mente e utilizza l'algoritmo di triturazione e assemblaggio dei record descritto nel documento Dremel. Riteniamo che questo approccio sia superiore al semplice appiattimento degli spazi dei nomi annidati.

Parquet è progettato per supportare schemi di compressione e codifica molto efficienti. Più progetti hanno dimostrato l'impatto sulle prestazioni dell'applicazione del giusto schema di compressione e codifica ai dati. Parquet consente di specificare gli schemi di compressione a livello di colonna, ed è a prova di futuro per consentire l'aggiunta di ulteriori codifiche man mano che vengono inventate e implementate.

Parquet è raccomandato il formato file con impala Tables nelle distribuzioni Cloudera.

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

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

AVRO

I file Avro sono stati supportati in Hive 0.14.0 e versioni successive.

Avro è una chiamata di procedura remota e un framework di serializzazione dei dati sviluppato all'interno del progetto Hadoop di Apache. Usa JSON per definire tipi di dati e protocolli e serializza i dati in un formato binario compatto. Il suo uso principale è in Apache Hadoop, dove può fornire sia un formato di serializzazione per i dati persistenti, sia un formato wire per la comunicazione tra i nodi Hadoop e dai programmi client ai servizi Hadoop.

Specifica del formato 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');
```

Possiamo anche utilizzare la sintassi sottostante senza utilizzare il file di schema.

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

Negli esempi sopra `STORED AS AVRO` clausola `STORED AS AVRO` è equivalente a:

```
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'
```

File di testo

TextFile è il formato file predefinito, a meno che il parametro di configurazione hive.default.fileformat abbia un'impostazione diversa. Possiamo creare una tabella su hive usando i nomi dei campi nel nostro file di testo delimitato. Diciamo per esempio, il nostro file csv contiene tre campi (id, nome, stipendio) e vogliamo creare una tabella in hive chiamata "dipendenti". Utilizzeremo il seguente codice per creare la tabella nell'alveare.

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

Ora possiamo caricare un file di testo nella nostra tabella:

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

Visualizzare i contenuti della nostra tabella su hive per verificare se i dati sono stati caricati correttamente:

```
SELECT * FROM employees;
```

Leggi Formati di file in HIVE online: https://riptutorial.com/it/hive/topic/4513/formati-di-file-in-hive

Capitolo 6: Funzioni aggregate definite dall'utente (UDAF)

Examples

Esempio medio UDAF

- Creare una classe Java che estenda `org.apache.hadoop.hive.ql.exec.hive.UDAF` Crea una classe interna che implementa `UDAFEvaluator`
- Implementa cinque metodi
    - `init()` - Questo metodo inizializza il valutatore e ripristina il suo stato interno. Stiamo usando la nuova colonna () nel codice qui sotto per indicare che nessun valore è stato ancora aggregato.
    - `iterate()` - Questo metodo viene chiamato ogni volta che è presente un nuovo valore da aggregare. Il valutatore dovrebbe aggiornare il suo stato interno con il risultato di eseguire l'aggregazione (stiamo facendo la somma - vedi sotto). Restituiamo true per indicare che l'input era valido.
    - `terminatePartial()` - Questo metodo viene chiamato quando Hive vuole un risultato per l'aggregazione parziale. Il metodo deve restituire un oggetto che incapsula lo stato dell'aggregazione.
    - `merge()` - Questo metodo viene chiamato quando Hive decide di combinare un'aggregazione parziale con un'altra.
    - `terminate()` - Questo metodo viene chiamato quando è necessario il risultato finale dell'aggregazione.

```
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;
```

Leggi Funzioni aggregate definite dall'utente (UDAF) online:
https://riptutorial.com/it/hive/topic/5137/funzioni-aggregate-definite-dall-utente--udaf-

Capitolo 7: Funzioni della tabella definite dall'utente (UDTF)

Examples

Esempio e utilizzo UDTF

Funzioni della tabella definite dall'utente rappresentate dall'interfaccia **org.apache.hadoop.hive.ql.udf.generic.GenericUDTF** . Questa funzione consente di generare più righe e più colonne per un singolo input.

Dobbiamo sovrascrivere i seguenti metodi:

```
 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
      }
}
```

Imballa il codice in jar e devi aggiungere jar al contesto hive.

```
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
```

Leggi Funzioni della tabella definite dall'utente (UDTF) online:
https://riptutorial.com/it/hive/topic/6502/funzioni-della-tabella-definite-dall-utente--udtf-

Capitolo 8: Hive User Defined Functions (UDF's)

Examples

Creazione UDF di Hive

Per creare una UDF, dobbiamo estendere la classe UDF (`org.apache.hadoop.hive.ql.exec.UDF`) e implementare il metodo di valutazione.

Una volta che UDF è stato rispettato e il JAR è stato creato, è necessario aggiungere jar al contesto hive per creare una funzione temporanea / 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 per tagliare la stringa data.

```
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;
 }
}
```

esportare il file sopra in jar

Vai alla CLI di Hive e aggiungi l'UDF JAR

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

Verificare che JAR si trovi in Hive CLI Classpath

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

Crea funzione temporanea

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

Uscita UDF

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

Leggi Hive User Defined Functions (UDF's) online: https://riptutorial.com/it/hive/topic/4949/hive-user-defined-functions--udf-s-

Capitolo 9: indicizzazione

Examples

Struttura

```
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 (...)]
```

Esempio:

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

Alter Index

ALTER INDEX nome_indice ON nome_tabella [PARTITION (...)] REBUILD

Drop Index

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

Se WITH REBUILD DEFERRED è specificato in CREATE INDEX, l'indice appena creato è inizialmente vuoto (indipendentemente dal fatto che la tabella contenga dati).

Il comando ALTER INDEX REBUILD può essere utilizzato per creare la struttura dell'indice per tutte le partizioni o una singola partizione.

Leggi indicizzazione online: https://riptutorial.com/it/hive/topic/6365/indicizzazione

Capitolo 10: Inserisci istruzione

Sintassi

- **Sintassi standard:**
- INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [SE NON ESISTE]] select_statement1 FROM from_statement;
- INSERT INTO TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] select_statement1 FROM from_statement;
- INSERIRE NELLA TABELLA tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] (z, y) select_statement1 FROM from_statement;
- **Estensione dell'alveare (più inserti):**
- FROM from_statement
  INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [SE NON ESISTE]] select_statement1
  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [SE NON ESISTE]] select_statement2]
  [INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2] ...;
- FROM from_statement
  INSERIRE NELLA TABELLA tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] select_statement1
  [INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2]
  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [SE NON ESISTE]] select_statement2] ...;
- **Estensione Hive (inserimenti di partizioni dinamiche):**
- INSERT OVERWRITE TABLE tablename PARTITION (partcol1 [= val1], partcol2 [= val2] ...) select_statement FROM da_statement;
- INSERIRE NELLA TABELLA tablename PARTITION (partcol1 [= val1], partcol2 [= val2] ...) select_statement FROM from_statement;

Osservazioni

inserire la sovrascrittura
Un'istruzione di sovrascrittura dell'inserto elimina tutti i file esistenti nella tabella o partizione di destinazione prima di aggiungere nuovi file in base all'istruzione select utilizzata. Si noti che quando ci sono modifiche alla struttura di una tabella o al DML utilizzato per caricare la tabella, a volte i vecchi file non vengono cancellati. Quando si carica su una tabella utilizzando il

partizionamento dinamico, solo le partizioni definite dall'istruzione select verranno sovrascritte. Tutte le partizioni preesistenti nel target rimarranno e non verranno eliminate.

inserire
Un inserimento nell'istruzione aggiunge nuovi dati in una tabella di destinazione in base all'istruzione select utilizzata.

Examples

inserire la sovrascrittura

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

Inserisci nella tabella

INSERT INTO verrà aggiunto alla tabella o alla partizione, mantenendo intatti i dati esistenti.

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

Se una tabella è partizionata, possiamo inserire in quella particolare partizione in modo statico come mostrato di seguito.

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

Se una tabella è partizionata, possiamo inserire in quella particolare partizione in modo dinamico come mostrato di seguito. Per realizzare inserti di partizione dinamici dobbiamo impostare sotto le proprietà.

```
    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;
```

Inserimenti multipli da una tabella.

Estensione dell'alveare (più inserti):

```
    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'
```

Leggi Inserisci istruzione online: https://riptutorial.com/it/hive/topic/1744/inserisci-istruzione

Capitolo 11: Script di creazione tabella con dati di esempio

Examples

Tipi di data e timestamp

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

Valori di dati minimi e massimi:

```
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');
```

Tipi di testo

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

Dati di esempio:

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

Tipi numerici

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

Valori di dati minimi e massimi:

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

Tipi numerici a virgola mobile

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

Valori di dati minimi e massimi:

```
insert into all_floating_numeric_types values (-3.4028235E38,-1.79769313486231570E308);
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 (3.4028235E38,1.79769313486231570E308);
```

Tipi booleani e binari

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

Dati di esempio:

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

Nota:

- Per booleano, internamente è memorizzato come vero o falso.
- Per binario, memorizzerà il valore codificato base64.

Tipi complessi

ARRAY

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

Crea `data.csv` con i dati:

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

Inserisci `data.csv` nella cartella `/tmp` e carica questi dati in Hive

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

Oppure puoi inserire questo CSV in HDFS dire in `/tmp` . Carica i dati da CSV su HDFS usando

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

CARTA GEOGRAFICA

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

file `data.csv` :

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

Carica dati nell'alveare:

```
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 '&';
```

file `data.csv` :

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

Carica dati nell'alveare:

```
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 '&';
```

file `data.csv` :

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

Carica dati nell'alveare:

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

Leggi Script di creazione tabella con dati di esempio online:
https://riptutorial.com/it/hive/topic/5067/script-di-creazione-tabella-con-dati-di-esempio

Capitolo 12: SELECT Statement

Sintassi

- SELEZIONA [TUTTI | DISTINCT] select_expr, select_expr, select_expr, ...
- Da table_reference
- [WHERE where_condition]
- [GROUP BY col_list]
- [AVENDO di avere la condizione]
- [ORDER BY col_list]
- [LIMIT n]

Examples

Seleziona tutte le righe

`SELECT` viene utilizzato per recuperare righe di dati da una tabella. Puoi specificare quali colonne verranno recuperate:

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

Oppure usa * per ottenere tutte le colonne:

```
SELECT *
FROM Employees;
```

Seleziona righe specifiche

Questa query restituirà tutte le colonne dalla tabella `sales` in cui i valori nella colonna `amount` è maggiore di 10 ed i dati nella `region` colonna a "USA".

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

Puoi usare *le espressioni regolari* per selezionare le colonne che vuoi ottenere. La seguente dichiarazione otterrà i dati dal `name` colonna e tutte le colonne che iniziano con l' `address` prefisso.

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

Puoi anche utilizzare la parola chiave `LIKE` (combinata con il carattere '%') per far corrispondere le stringhe che iniziano con o terminano con una sottostringa particolare. La seguente query restituirà tutte le righe in cui la `city` della colonna inizia con "Nuovo"

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

È possibile utilizzare la parola chiave `RLIKE` per utilizzare *le espressioni regolari* Java. La seguente query restituirà le righe il cui `name` colonna contiene le parole "smith" o "figlio".

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

È possibile applicare le funzioni ai dati restituiti. La seguente frase restituirà tutto il nome in maiuscolo.

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

È possibile utilizzare diverse funzioni matematiche , funzioni di raccolta , funzioni di conversione del tipo , funzioni di data , funzioni condizionali o funzioni di stringa .

Per limitare il numero di righe indicate nel risultato, è possibile utilizzare la parola chiave `LIMIT` . La seguente dichiarazione restituirà solo dieci righe.

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

Seleziona: Progetto colonne selezionate

Tabella di esempio (struttura Dipendente)

Nome colonna	Tipo di dati
ID	INT
F_Name	STRINGA
L_Name	STRINGA
Telefono	STRINGA
Indirizzo	STRINGA

Proietta tutte le colonne

Usa jolly `*` per proiettare tutte le colonne. per esempio

```
Select * from Employee
```

Colonne selezionate progetto (ad esempio ID, nome)

Usa il nome delle colonne nell'elenco di proiezione. per esempio

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

Scarta 1 colonna dall'elenco Proiezione

Mostra tutte le colonne tranne 1 colonna. per esempio

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

Elimina le colonne corrispondenti al modello

Rifiuta tutte le colonne che corrispondono al modello. ad es. Rifiuta tutte le colonne che terminano con `NAME`

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

Leggi SELECT Statement online: https://riptutorial.com/it/hive/topic/4133/select-statement

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con l'alveare	Bhavesh, Community, franklinsijo, johnnyaug, NeoWelkin
2	Crea un database e una tabella	CodingInCircles, dev ツ, goks, Panther, Venkata Karthik
3	Creazione di tabelle alveare tramite Sqoop	NeoWelkin
4	Esportare i dati in Hive	Prem Singh Bist
5	Formati di file in HIVE	agentv, Alex, Community, johnnyaug, leftjoin, Mzzzzzz, NeoWelkin, tomek, Venkata Karthik
6	Funzioni aggregate definite dall'utente (UDAF)	dev ツ, Venkata Karthik
7	Funzioni della tabella definite dall'utente (UDTF)	Venkata Karthik
8	Hive User Defined Functions (UDF's)	Ashok, Panther, Venkata Karthik
9	indicizzazione	Prem Singh Bist
10	Inserisci istruzione	Jared, johnnyaug, Venkata Karthik
11	Script di creazione tabella con dati di esempio	dev ツ
12	SELECT Statement	Ambrish, dev, Jaime Caffarel, johnnyaug