無料電子ブック

学習

hive

Free unaffiliated eBook created from
Stack Overflow contributors.

#hive

1: ハイブをめる

- Hiveは、 Hadoopのにされたデータウェアハウスツールです。
- データをするためのSQLライクなをします。
- ほとんどのSQLクエリはHiveでできますが、のいは、バックエンドでmap-reduceジョブをしてHadoop Clusterからをすることです。このため、Hiveはセットをするのにがかかることがあります。

Examples

ハイブのの

ドキュメントファイルファイル

メアリーはさなをっていた

そのフリースはのようにかった

メアリーがどこにっても

はかにきました。

ハイブクエリ

```
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;
```

Hiveの**word_counts**テーブルの

メアリー、2

っていた、1

a、1

し、1

、2

その、1

フリーース、1

だった、2

ホワイト、1

as、1

、1

そして、1

どこでも、1

それ、1

った、1

、1

かに、1

、1

く、1

Hivelinuxのインストール

https://hive.apache.org/downloads.htmlからのをダウンロードすることからめます。

- >すぐファイルをuntarしてください

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

- > / usr / local /にディレクトリをする

$ sudo mkdir / usr / local / hive

- >ファイルをrootにする

$ mv/ Downloads / hive-2.xy / usr / local / hive

- > hadoopとハイブのを.bashrcでする

$ gedit/ .bashrc

このような

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

export PATH = $ PATH$ HIVE_HOME / bin

エクスポートCLASSPATH = $ CLASSPATH/ usr / local / Hadoop / lib / *。

エクスポートCLASSPATH = $ CLASSPATH/usr/local/hive/apache-hive-2.0.1-bin/lib/* :.

- >さて、まだされていないは、hadoopをしてください。それがされていることをし、セーフモードではありません。

$ hadoop fs -mkdir / user / hive / warehouse

ディレクトリ「ウェアハウス」は、ハイブにするテーブルまたはデータをするです。

$ hadoop fs -mkdir / tmp

ディレクトリ "tmp"は、のをするなです。

- >これらのフォルダにするみり/きみのをします。

$ hadoop fs -chmod g +ユーザー/ハイブ/

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

→はコンソールでこのコマンドをってHIVEをしてください

$ハイブ

LinuxでのMetastoreによるハイブインストール

1. Java 7
2. HadoopHadoopのインストールについてはここをしてください
3. MySQLのサーバーとクライアント

インストール

ステップ1ダウンロードページからのハイブタールをダウンロードします。

ステップ2ダウンロードしたtarballをする *$ HOME*にダウンロードする

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

ステップ3ファイル `~/.bashrc` をする

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

しいをするためにファイルをソースします。

```
source ~/.bashrc
```

ステップ4mysqlのJDBCコネクタをここからダウンロードしてします。

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

されたディレクトリには、コネクタjarファイル`mysql-connector-java-abcjar`がまれています。`$HIVE_HOME lib`にコピーして`$HIVE_HOME`

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

`$HIVE_HOME/conf/` directoryのにハイブファイル`$HIVE_HOME/conf/ hive-site.xml`し、のメタストアプロパティをします。

```
<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>
```

MySQLの "username"と "password"のをプロパティにじてします。

メタストアスキーマの

メタストアスキーマスクリプトは、 `$HIVE_HOME/scripts/metastore/upgrade/mysql/`

MySQLにログインしてスキーマをソースし、

```
mysql -u username -ppassword

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

Metastoreの

```
hive --service metastore
```

バックグラウンドでするには、

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

HiveServer2のにじて

```
hiveserver2
```

バックグラウンドでするには、

```
nohup hiveserver2 metastore &
```

これらのファイルは`$HIVE_HOME/bin/`

`hive`にするには、 `hive` 、 `beeline`またはのいずれかをします。

Hive CLIはです。BeelineまたはHueをすることをおめします。

の

このを`$HUE_HOME/desktop/conf/hue.ini`

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

オンラインでハイブをめるをむ https://riptutorial.com/ja/hive/topic/1099/ハイブをめる

2: HIVEのファイル

Examples

SEQUENCEFILE

データをするがあるは、SEQUENCEFILEにデータをします。 GzipまたはBzip2でされたテキストファイルをTextFileとしてされたテーブルにインポートすることができます。はにされ、ファイルはクエリのにオンザフライでされます。

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

ORC

ORCOptimized Row Columnarファイルは、Hiveデータをにするをします。のHiveファイルのをするようにされています。 ORACファイルをすると、Hiveがデータのみり、きみ、およびにパフォーマンスがします。 ORCファイルにはインデックスとブルームフィルタをめることができます。

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

ORCは、HortonWorksディストリビューションのデータをするためのフォーマットです。

```
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"
              )
```

テーブルのしいパーティションがORCファイルとしてされるようにテーブルをするには

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

Hive 0.14、ユーザーはまたはパーティションで`CONCATENATE`コマンドをすることにより、さなORCファイルのなマージをできます。ファイルはスキャンせずにストライプレベルでマージされます。

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

PARQUET

Hive 0.13.0では、のストレージ。りは、なネストされたデータをにいてされ、Dremelのにされているレコードシュレッディングとアセンブリアルゴリズムをしています。このアプローチはネストされたのなよりれているとえています。

りは、になおよびをサポートするようにされています。のプロジェクトが、なおよびをデータにすることによるパフォーマンスのをしました。りでは、スキームをのレベルですることができ、には、されされたときにさらにくのエンコーディングをできるようになります。

りは、Clouderaのインパラテーブルをつファイルがされます。

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

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

AVRO

AvroファイルはHive 0.14.0でサポートされています。

Avroは、ApacheのHadoopプロジェクトでされたリモートプロシージャコールとデータのシリアルフレームワークです。 JSONをしてデータとプロトコルをし、コンパクトなバイナリでデータをシリアライズします。なはApache Hadoopです。ここでは、データのシリアルと、Hadoopノードの、およびクライアントプログラムからHadoopサービスへののためのワイヤのをできます。

AVROの 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');
```

スキーマファイルをせずにのをすることもできます。

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

の`STORED AS AVRO`のは、のものとです。

```
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'
```

テキストファイル

TextFileは、パラメータhive.default.fileformatのがなるをき、デフォルトのファイルです。られたテキストファイルのフィールドをって、ハイブにテーブルをすることができます。たとえば、csvファイルに3つのフィールドid、name、salaryがまれており、 "employees"というハイブでテーブルをしたいとします。のコードをしてハイブにテーブルをします。

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

これでテーブルにテキストファイルをみむことができます

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

データがにみまれたかどうかをするためにハイブにテーブルのをする

```
SELECT * FROM employees;
```

オンラインでHIVEのファイルをむ https://riptutorial.com/ja/hive/topic/4513/hiveのファイル

3: SELECTステートメント

- SELECT [すべて| DISTINCT] select_expr、select_expr、select_expr、...。
- FROM table_reference
- [WHERE where_condition]
- [GROUP BY col_list]
- [き]
- [ORDER BY col_list]
- [LIMIT n]

Examples

すべてのを

`SELECT`は、テーブルからデータのをりすためにされます。するをできます。

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

または、*をしてすべてのをするだけです。

```
SELECT *
FROM Employees;
```

のを

このクエリは、`amount`が10よりきい「`sales`テーブルのすべてのと、「US」の`region`のデータをします。

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

をして、するをすることができます。のステートメントは、`name`と`address`まるすべてのからデータをし`address`。

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

キーワード`LIKE` "とみわされたをして、のでまるまたはそのでわるをすることもできます。のクエリは、`city`が「」でまるすべてのをします。

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

`RLIKE`キーワードをすると、Javaのをできます。のクエリは、`name`に "smith"または "son"というがまれているをします。

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

されたデ―タにをすることができます。のはすべてのをでします。

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

さまざまな 、 コレクション 、 、 、 、またはをできます 。

resultにえられたのをするために、 `LIMIT`キ―ワ―ドをすることができます。のは10だけをします。

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

プロジェクトをした

サンプルテ―ブル**Employee**

	データ.タイプ
ID	INT
F_Name	STRING
L_Name	STRING
	STRING
	STRING

すべてのをする

ワイルドカ―ド`*`をしてすべてのをします。えば

```
Select * from Employee
```

プロジェクトをした**ID**、など

リストののをします。えば

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

プロジェクションリストから**1**をする

1をくすべてのをします。えば

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

パターンにするをする

パターンにするすべてのをします。 NAMEわるすべてのをする

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

オンラインでSELECTステートメントをむ https://riptutorial.com/ja/hive/topic/4133/selectステートメント

4: Sqoopによるハイブテーブルの

き

たちのHDFSクラスタにけられたHiveメタストアがある、Sqoopは、Hiveでデータのレイアウトをするcreate tableステートメントをしてすることによって、データをHiveにインポートできます。 Hiveへのデータのインポートは、Sqoopコマンドラインに--hive-importオプションをするのとじくらいです。

RDBMSからHIVEにデータをインポートすると、くのをできます。また、のクエリまたはなクエリをし、したテーブルにHiveにすることもできます。

> --hive-importはSqoopになHiveでHDFSではないことをえます。
>
> --hive-tableオプションは、たちがしたハイブのテーブルにデータをインポートするのにちます。それのは、RDBMSからインポートされるソーステーブルのになります。

Examples

ハイブのDestinationテーブルをむHive インポート

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

オンラインでSqoopによるハイブテーブルのをむ https://riptutorial.com/ja/hive/topic/10685/sqoopによるハイブテーブルの

5: インデックス

Examples

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

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

インデックスをする

ALTER INDEX index_name ONテーブル[PARTITION...] REBUILD

ドロップインデックス

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

WITH DEFERRED REBUILDがCREATE INDEXでされている、しくされたははですにデータがまれているかどうかはありません。

ALTER INDEX REBUILDコマンドをすると、すべてのパーティションまたはパーティションのをできます。

オンラインでインデックスをむ https://riptutorial.com/ja/hive/topic/6365/インデックス

6: サンプルデータをむテーブルスクリプト

Examples

とタイムスタンプの

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

と

```
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');
```

テキストタイプ

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

サンプルデータ

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

タイプ

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

と

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

```
CREATE TABLE all_floating_numeric_types(
```

```
  c_float float,
  c_double double
);
```

と

```
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);
```

ブールとバイナリ

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

サンプルデータ

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

- ブールの、にtrueまたはfalseとしてされます。
- バイナリの、base64でエンコードされたがされます。

アレイ

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

データきの`data.csv`をする

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

`data.csv`を`/tmp`フォルダにき、このデータをハイブにロードする

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

または、このCSVを`/tmp` HDFSにれることもできます。 HDFSでのCSVからのデータのロード

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

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

data.csvファイル

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

データをハイブにロードする

```
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 '&';
```

data.csvファイル

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

データをハイブにロードする

```
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 '&';
```

data.csvファイル

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

データをハイブにロードする

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

オンラインでサンプルデータをむテーブルスクリプトをむ
https://riptutorial.com/ja/hive/topic/5067/サンプルデータをむテーブルスクリプト

7: データベースとテーブルステートメントの

- CREATE [TEMPORARY] [EXTERNAL] TABLE [しない] [db_name。] table_name

  [col_name data_type [COMMENT col_comment]、...] [COMMENT table_comment] [PARTSTIONED BYcol_name data_type [COMMENT col_comment]、...] [CLUSTERED BY col_name、col_name、... Hive 0.10.0で] ONcol_value、col_value、col_value、col_value、col_value、col_value、 ...、col_value、col_value、...、...[DORECTORIESとしてされている] [[ROW FORMAT row_format] [STORED AS file_format] | STORED BY 'storage.handler.class.name' [SERDEPROPERTIES...]] [ロケーションhdfs_path] [TBLPROPERTIESproperty_name = property_value、...]
  [AS select_statement];

- CREATE [TEMPORARY] [EXTERNAL] TABLE [しない] [db_name。] table_name LIKE existing_table_or_view_name [LOCATION hdfs_path];

- data_typeprimitive_type、array_type、map_type、struct_type、union_type

- プリミティブTINYINT、SMALLINT、INT、BIGINT、BOOLEAN、FLOAT、DOUBLE、STRING、BINARY、TIMESTAMP、DECIMAL、DECIMAL、り、DATE、VARCHAR、CHAR

- array_typeARRAY <データ>

- map_typeMAP <primitive_type、data_type>

- struct_typeSTRUCT <col_namedata_type [コメントcol_comment]、...>

- union_typeUNIONTYPE <データ、データ、...>

- char [ESCAPED BY char] [FILLDS TERMINATED BY char [ESCAPED BY char]] [charでわるコレクションアイテム] [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char] [NULL DEFINED AS char]
  、SERDE serde_name [SERDEPROPERTIESプロパティ=プロパティ、プロパティ=プロパティ、...]

- file_format :: SEQUENCEFILE、TEXTFILE、RCFILE、ORC、PARQUET、AVRO、INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

- CREATEデータベース|スキーマ[しない] database_name [コメントデータベース_コメント] [ロケーションhdfs_path] [WITH DBPROPERTIESproperty_name = property_value、...];

HIVEでテーブルとデータベースをするとき。のがです。

- `use database;`を`use database;`りえることができ`use database;`コマンド
- `SELECT current_database()`

をしてできるのデータベースをるには、
- create tableにされるDDLをするには、 `SHOW CREATE TABLE tablename`
- テーブルのすべてのをするには、 `DESCRIBE tablename`をして、されたロケーションserdeや `DESCRIBE FORMATTED tablename`などのをし`DESCRIBE FORMATTED tablename`。 DESCRIBEはDESCとすることもできます。

Examples

テーブルの

パーティションきのテーブルをし、シーケンスファイルとしてします。ファイルのデータは、`Ctrl-A (^A)`でフィールドりとでられているとされています。のは、Hiveファイル `hive.metastore.warehouse.dir hive-site.xml`のキー`hive.metastore.warehouse.dir`にされているハイブウェアハウスディレクトリにされています。

```
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;
```

パーティションをつテーブルをし、シーケンスファイルとしてします。ファイルのデータフォーマットは、 `ctrl-A`でフィールドりとでりにするとしています。のはされたにされており、にデータがあるにです。テーブルをするの1つは、データをせずにテーブルをできることです。たとえば、テーブルをしてスキーマがっているとわかったら、データをにせずにテーブルをとしてしいスキーマですることができます。のは、じファイルにのようなツールをしている、テーブルをしてもききすることができます。

```
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>';
```

クエリをしてテーブルをし、クエリのをすると、これらのステートメントは**CTAS**テーブルをとしてとばれます 。

CTASには2つのがあり、SELECTはHiveQLでサポートされているSELECTです。 CTASのCREATEは、としてされたスキーマをSELECTからりし、SerDeやなどののプロパティをつターゲットをします。

CTASにはのがあります。

- ターゲットは、パーティションにすることはできません。
- ターゲットをにすることはできません。
- ターゲットテーブルはリストバケットテーブルにすることはできません。

```
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;
```

テーブルをする

LIKEの**CREATE TABLE**をすると、のテーブルをデータをコピーせずににコピーすることができます。 CTASとはに、のステートメントは、テーブルのすべてののテーブルのとにするしいテーブルをします。しいテーブルにははまれません。

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

データベースの

のにデータベースをする。々は、データベースのをするのではなく、ウェアハウスディレクトリにします。

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

ハイブ**ACID**テーブルの。

ACIDテーブルはハイブ0.14サポートされています。のは、UPDATE / DELETE / INSERTをサポートしています。

hive-site.xmlになの

```
 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
```

では、orcファイルのみがサポートされています。

テーブルステートメント。

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

HIVE_HBASE

Hive-Hbaseは、のバージョンでサポートされています。ハイブ0.11.0Haase0.94.2Hadop0.20.2

```
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");
```

1はキーでなければなりません。

のテーブルプロパティをしてテーブルをします。

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

オンラインでデータベースとテーブルステートメントのをむ
https://riptutorial.com/ja/hive/topic/3328/データベースとテーブルステートメントの

8: ハイブでのデータのエクスポート

Examples

ハイブのエクスポート

テーブルから/ **tmp / ca_employees**へのデータのエクスポート

OVERWRITE LOCAL DIRECTORY '/ tmp / ca_employees' SELECT、、のse.state = 'CA';

のについてテーブルからのローカルディレクトリにデータをエクスポートする

のクエリは、のについてのディレクトリにデータをエクスポートするためにのをするをしています

```
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';
```

オンラインでハイブでのデータのエクスポートをむ https://riptutorial.com/ja/hive/topic/6530/ハイブでのデータのエクスポート

9: ハイブユーザーUDF

Examples

ハイブUDFの

UDFをするには、UDF `org.apache.hadoop.hive.ql.exec.UDF` クラスをし、メソッドをするがあります。

UDFがし、JARがビルドされたら、/なをするために、hiveコンテキストにjarをするがあります。

```
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;
```

されたをトリミングするハイブUDF。

```
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;
 }
}
```

をjarファイルにエクスポートする

Hive CLIにし、UDF JARをする

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

JARがハイブCLIクラスパスにあることをする

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

をする

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

UDF

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

オンラインでハイブユーザーUDFをむ https://riptutorial.com/ja/hive/topic/4949/ハイブユーザー-udf-

10: ユーザーテーブルUDTF

Examples

UDTFの

org.apache.hadoop.hive.ql.udf.generic.GenericUDTFインターフェースでされるユーザーの。
このをすると、のにしてのとのをできます。

のメソッドをきするがあります

```
 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
      }
}
```

コードをjarファイルにパッケージし、jarをhiveコンテキストにするがあります。

```
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
```

オンラインでユーザーテーブルUDTFをむ https://riptutorial.com/ja/hive/topic/6502/ユーザーテーブル-udtf-

11: ユーザーUDAF

Examples

UDAFの

- `org.apache.hadoop.hive.ql.exec.hive.UDAF` `UDAFEvaluator`するJavaクラスをする`UDAFEvaluator`をするクラスをする
- 5つのメソッドをする
  - `init()` - このメソッドはをし、をリセットします。のコードでしいColumnをして、がまだされていないことをします。
  - `iterate()` - このメソッドは、するしいがあるたびにびされます。は、をしたでをするがありますをっています - を。がであることをすためにtrueをします。
  - `terminatePartial()` - このメソッドは、Hiveがのをとするときにびされます。このメソッドは、のをカプセルするオブジェクトをすがあります。
  - `merge()` - このメソッドは、Hiveが1つのをのとすることをしたときにびされます。
  - `terminate()` - このメソッドは、のがなときにびされます。

```
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;
```

オンラインでユーザーUDAFをむ https://riptutorial.com/ja/hive/topic/5137/ユーザー-udaf-

12: をする

-
- きテーブルのtablename1 [PARTITIONpartcol1 = val1、partcol2 = val2 ...[IF NOT EXISTS]] select_statement1 FROM from_statement;
- INSERT INTO TABLE tablename1 [PARTITIONpartcol1 = val1、partcol2 = val2 ...] select_statement1 FROM from_statement;
- INSERT INTO TABLE tablename1 [PARTITIONpartcol1 = val1、partcol2 = val2 ...]z、y select_statement1 FROM from_statement;
- ハイブの
- FROM from_statement
  きテーブルのtablename1 [PARTITIONpartcol1 = val1、partcol2 = val2 ...[しない]] select_statement1
  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [しない]] select_statement2]
  [INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2] ...;
- FROM from_statement
  INSERT INTO TABLE tablename1 [PARTITIONpartcol1 = val1、partcol2 = val2 ...] select_statement1
  [INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2]
  [きテーブルをするtablename2 [パーティション... [しない]] select_statement2] ...;
- ハイブパーティション
- INSERT OVERWRITE TABLE tablename PARTITIONpartcol1 [= val1]、partcol2 [= val2] ... select_statement FROM from_statement;
- INSERT INTO TABLE tablename PARTITIONpartcol1 [= val1]、partcol2 [= val2] ... select_statement FROM from_statement;

きをする

insert overwriteは、されるselectについてしいファイルをするに、ターゲットまたはパーティションののファイルをします。またはをロードするためにされるDMLにがあって、いファイルがされないことがあることにしてください。パーティションをしてテーブルにロードする、selectステートメントでされたパーティションだけがきされます。ターゲットののパーティションはそのまま、されません。

する

insert intoは、されるselectについてしいデータをターゲットにします。

Examples

きをする

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

テーブルに

INSERT INTOはテーブルまたはパーティションにし、のデータをそのまま します。

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

がパーティションされている、そののパーティションにのようににできます。

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

テーブルがされている、そののパーティションににすることができます。パーティションをするには、のプロパティのにするがあります。

```
    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;
```

テーブルからのの

ハイブの

```
    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'
```

オンラインでをするをむ https://riptutorial.com/ja/hive/topic/1744/をする

クレジット

S. No		Contributors
1	ハイブをめる	Bhavesh, Community, franklinsijo, johnnyaug, NeoWelkin
2	HIVEのファイル	agentv, Alex, Community, johnnyaug, leftjoin, Mzzzzzz, NeoWelkin, tomek, Venkata Karthik
3	SELECTステートメント	Ambrish, dev, Jaime Caffarel, johnnyaug
4	Sqoopによるハイブテーブルの	NeoWelkin
5	インデックス	Prem Singh Bist
6	サンプルデータをむテーブルスクリプト	dev ツ
7	データベースとテーブルステートメントの	CodingInCircles, dev ツ, goks, Panther, Venkata Karthik
8	ハイブでのデータのエクスポート	Prem Singh Bist
9	ハイブユーザーUDF	Ashok, Panther, Venkata Karthik
10	ユーザーテーブルUDTF	Venkata Karthik
11	ユーザーUDAF	dev ツ, Venkata Karthik
12	をする	Jared, johnnyaug, Venkata Karthik