==========SAMPLE==========

NASDAQ,JDAS,2010-01-29,26.91,27.53,26.02,26.21,883100,26.21

CREATE TABLE STOCK_HIVE (EXCHNGE STRING,SYMBOL STRING,DATE STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT,ADJ_CLOSE DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

load data local inpath '/home/user/work/input/StockData' into table stock_hive;

SELET OPEN AS OPEN, CLOSE AS CLOSE FROM STOCK_HIVE WHERE HIGH >10;

CREATE EXTERNAL TABLE STOCK_HIVE_EXTERANAL (EXCHNGE STRING,SYMBOL STRING,DATE STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT,ADJ_CLOSE DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/hdfs/input/StockData/';

CREATE TABLE students_bucket(name STRING,id INT,college STRING) PARTITIONED BY(country STRING) CLUSTERED BY (college) INTO 4 BUCKETS ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ;

veera 101 gsdc

ravi 102 vrc

raj 103 vrc

nayan 104 gkg

satish 105 bgr

mahee 106 gsdc

==========SAMPLE END==========

****

MANAGED TABLE

****

CREATE DATABASE retail_db;

CREATE DATABASE IF NOT EXISTS retail_db;

USE retail_db;

DROP TABLE categories;

```
CREATE TABLE categories (
category_id int, category_department_id int,
category_name string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

== The table is created in default location(/user/hive/warehouse/hivetest.db/categories/)

== load local data into hive table

```
LOAD DATA LOCAL INPATH '/usr/local/localinput/data/categories/categories.csv' into table categories;
```

== load hdfs data into hive table

```
LOAD DATA INPATH '/training1/categories.csv' into table categories;
```

== The data file will moved to /user/hive/warehouse/hivetest.db/categories/

```
CREATE TABLE customers (
customer_id       int, customer_fname    string,
customer_lname    string, customer_email    string,
customer_password string, customer_street   string,
customer_city     string, customer_state    string,
customer_zipcode  string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

==========

```
CREATE TABLE departments (
department_id int, department_name string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

==========

```
CREATE TABLE orders (
order_id int, order_date string,
order_customer_id int, order_status string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

==========

```
CREATE TABLE order_items (
order_item_id int, order_item_order_id int,
order_item_order_date string, order_item_product_id int,
order_item_quantity smallint, order_item_subtotal float,
order_item_product_price float
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

==========

```
CREATE TABLE products (
product_id int,  product_category_id int,
product_name string, product_description string,
product_price float, product_image string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

****

PROPERTIES TO SEE CURRENT DATABASE AND TABLE HEADER :

****

```
set hive.cli.print.header=true;
set hive.cli.print.current.db=true;
```

****

MANAGED TABLE USING LOCATION : Data will not will not be move and if we drop table it will remove data from hdfs aswell.

****

```
CREATE TABLE categories_location (
category_id int, category_department_id int,
category_name string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE  LOCATION '/training1/categories/';
```

****

TEMPORARY TABLE : The table will exists till the session available..if you come out from hive prompt, table will be removed

****

```
CREATE TEMPORARY TABLE categories_temp (

category_id int, category_department_id int,

category_name string

) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

STORED AS TEXTFILE;
```

****

EXTERNAL TABLE USING LOCATION

****

```
CREATE EXTERNAL TABLE categories_extrnal_location (

category_id int,category_department_id int,

category_name string

)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

STORED AS TEXTFILE LOCATION '/training1/externalTable/categories'
```

********* example 2**********

```
CREATE EXTERNAL TABLE IF NOT EXISTS STUDENT_EXTERNAL1 (NAME STRING,ID DOUBLE,COLLEGE_NAME STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/input/hive_extrn_veera';

SELECT * FROM STUDENT_EXTERNAL;

CREATE EXTERNAL TABLE EXTERNAL_TEST3 (EXCHNGE STRING,SYMBOL STRING,DATE STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT,ADJ_CLOSE DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/INPUT/STOCKDATA.TXT';
```

****

EXTERNAL TABLE WITHOUT LOCATION :

****

if we did not specify the location, the table will get create under /user/hive/warehouse/ and it will treat as manage table. but when we drop it, it will delete the only table and schema not the data in hdfs.

```
CREATE EXTERNAL TABLE categories_extrnal_no_location (

category_id int,category_department_id int,

category_name string

)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

LOAD DATA LOCAL INPATH '/usr/local/localinput/data/categories/categories.csv' into table categories_extrnal_no_location;

**** Example 2**********

CREATE EXTERNAL TABLE EXTERNAL_TEST2 (EXCHNGE STRING,SYMBOL STRING,DATE STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT,ADJ_CLOSE DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

load data local inpath '/home/user/work/input/StockData.txt' into table external_test2;

-- LOAD DATA WHICH IS THERE IN HDFS.

when we load data from hdfs to hive table, it will move from hdfs to hive warehouse.

load data inpath '/hdfs/input/StockData/stock.txt' into table exteranal2;

CREATE TABLE EXTERNAL_TEST2 (EXCHNGE STRING,SYMBOL STRING,DATE STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT,ADJ_CLOSE DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

==========

SAME DATA WITH MULTIPLE SCHEMA

==========

CREATE EXTERNAL TABLE EXTERNAL_MULT_SCHEMA1 (EXCHNGE STRING,SYMBOL STRING,DATE STRING,OPEN DOUBLE,ADJ_CLOSE DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/input/StockData.txt';

CREATE EXTERNAL TABLE EXTERNAL_MULT_SCHEMA2 (EXCHNGE STRING,SYMBOL STRING,DATE STRING,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT,ADJ_CLOSE DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/input/StockData.txt';

==========

COMPLEX DATA TYPES

==========

====ARRAY DATATYPE======

SAMPLEDATA.TXT

EMIID,EMPNAME,SAL,ASSESTS,CITY

1,abc,40000,laptop$mouse$ph,hyd

2,def,3000,laptop$mouse,bang,

3,ravi,40000,abc,hyd

4,veera,3000,d$f,bang

CREATE TABLE TABLE_ARRAY(EMP_ID INT,EMP_NAME STRING,EMP_SAL BIGINT,ASSETS ARRAY<STRING>, CITY STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' COLLECTION ITEMS TERMINATED BY '$';

load data local inpath '/home/hadoop1/hiveinput/complexdata.txt' into table table_array;

hive> load data local inpath '/home/hadoop1/hiveinput/complexdata.txt' into table table_array;

Loading data to table veera_test.table_array

OK

Time taken: 1.21 seconds

hive> select * from table_array;

OK

1 abc 40000 ["a","b","c"] hyd

2 def 3000 ["d","f"] bang

3 ravi 40000 ["a","b","c"] hyd

4 veera 3000 ["d","f"] bang

Time taken: 2.458 seconds, Fetched: 4 row(s)

hive>select assets[1] from table_array where emp_id=1;

select assets[0] from table_array;

==========MAP DATATYPE ==========

Sample data

1,abc,40000,abc,pf#500$epf#200,hyd

2,def,3000,d$f,pf#500,bang

3,ravi,40000,abc,pf#500$epf#200,hyd

4,veera,3000,d$f,pf#500$epf#300$ppf#686,bang

CREATE TABLE TABLE_MAP(ID INT,NAME STRING,SAL BIGINT,ASSESTS ARRAY<STRING>,DEDUCTION MAP<STRING,INT>,CITY STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' COLLECTION ITEMS TERMINATED BY '$' MAP KEYS TERMINATED BY '#';

load data local inpath '/home/hadoop1/hiveinput/mapdata2.txt' into table table_map;

select deduction["pf"] from table_map;

select deduction["pf"],deduction["epf"] from table_map;

hive> load data local inpath '/home/hadoop1/hiveinput/mapdata2.txt' into table table_map;

Loading data to table veera_test.table_map

OK

Time taken: 0.392 seconds

hive> select * from table_map;

OK

1 abc 40000 ["a","b","c"] {"pf":500,"epf":200} hyd

2 def 3000 ["d","f"] {"pf":500} bang

3 ravi 40000 ["a","b","c"] {"pf":500,"epf":200} hyd

4 veera 3000 ["d","f"] {"pf":500,"epf":300,"ppf":686} bang

Time taken: 0.306 seconds, Fetched: 4 row(s)

hive> select deduction["pf"] from table_map;

OK

500

500

500

500

Time taken: 0.133 seconds, Fetched: 4 row(s)

hive> select deduction["pf"],deduction["epf"] from table_map;

OK

500 200

500 NULL

500 200

500 300

Time taken: 0.102 seconds, Fetched: 4 row(s)

==========STRUCT ==========

1,abc,40000,abc,pf#500$epf#200,hyd$ap$500001

2,def,3000,d$f,pf#500,bang$kar$600038

CREATE TABLE TABLE_STRUCT(ID INT,NAME STRING,SAL BIGINT,SUB ARRAY<STRING>,DUD MAP<STRING,INT>,ADDR STRUCT<CITY:STRING,STATE:STRING,PIN:BIGINT>) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' COLLECTION ITEMS TERMINATED BY '$' MAP KEYS TERMINATED BY '#';

load data local inpath '/home/hadoop1/hiveinput/stuctdata.txt' into table table_struct;

select addr.city from table_struct;

```
hive> load data local inpath '/home/hadoop1/hiveinput/stuctdata.txt' into table table_struct;

Loading data to table veera_test.table_struct

OK

Time taken: 0.383 seconds

hive> select * from table_struct;

OK

1	abc	40000	["a","b","c"]	{"pf":500,"epf":200}	{"city":"hyd","state":"ap","pin":500001}

2	def	3000	["d","f"]	{"pf":500}	{"city":"bang","state":"kar","pin":600038}

Time taken: 0.104 seconds, Fetched: 2 row(s)

hive> select addr.city from table_struct;

OK

hyd

bang

Time taken: 0.163 seconds, Fetched: 2 row(s)
```

==========

HIVE PARTITION EMPLOYEE -- STATIC PARTITION

==========

MANAGED TABLES PARTITION WITH LOAD COMMAND:

NOTE: WHEN GIVEN DATA DOES NOT HAVE PARTITION COLUMN, AND WE ARE GETTING THE DATA SEPARATELY BASED ON SOME PARAMETER LIKE CONUNTRY WISE

CASE-1 : THE DATA DOES NOT HAS PARTITION COLUMN.

```
CREATE TABLE IF NOT EXISTS STOCK_PARTITION_DATE_LOAD(SYMBOL STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT) PARTITIONED BY (STOCK_DATE STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/usr/local/hiveinput/staticpartition/NASDAQ_20161205.txt' INTO TABLE STOCK_PARTITION_DATE_LOAD PARTITION (STOCK_DATE='20161205');

LOAD DATA LOCAL INPATH '/usr/local/hiveinput/staticpartition/NASDAQ_20161206.txt' INTO TABLE STOCK_PARTITION_DATE_LOAD PARTITION (STOCK_DATE='20161206');

LOAD DATA LOCAL INPATH '/usr/local/hiveinput/staticpartition/NASDAQ_20161207.txt' INTO TABLE STOCK_PARTITION_DATE_LOAD PARTITION (STOCK_DATE='20161207');
```

========== EXAMPLE 2 ==========

CREATE TABLE EMP_PARTITION(ID STRING, NAME STRING, SAL BIGINT) PARTITIONED BY (COUNTRY STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

load data local inpath '/usr/local/localinput/partition/emp_in.txt' into table emp_partition partition (country='IN');

load data local inpath '/usr/local/localinput/partition/emp_us.txt' into table emp_partition partition (country='US');

load data local inpath '/usr/local/localinput/partition/emp_uk.txt' into table emp_partition partition (country='UK');

load data local inpath '/usr/local/localinput/partition/emp_uk.txt' into table emp_partition partition (country='IN');

========== EXAMPLE 3 ==========

CREATE TABLE IF NOT EXISTS STOCK_PARTITION(symbol string,stock_date string,open double,high double,low double,close double,volume bigint) PARTITIONED BY (country string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

load data local inpath '/usr/local/hiveinput/staticpartition/NASDAQ_20101105_IN.txt' into table stock_partition partition (country='IN');

load data local inpath '/usr/local/hiveinput/staticpartition/NASDAQ_20161207_US.csv' into table stock_partition partition (country='USA');

load data local inpath '/usr/local/hiveinput/staticpartition/NASDAQ_20101105_IN.txt' into table stock_partition partition (country='UK');

hive (test530)> select * from stock_partition where country='IN' LIMIT 10;

OK

hive (test530)> select * from stock_partition where country='UK' LIMIT 10;

OK

hive (test530)> select * from stock_partition where country='USA' LIMIT 10;

OK

==========EXAMPLE 4 MULTIPLE PARTITION==========

CREATE TABLE IF NOT EXISTS EMP_PARTITION(ID STRING, NAME STRING, SAL DOUBLE, DEPT STRING, DOJ STRING) PARTITIONED BY (COUNTRY STRING, STATE STRING, CITY STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH='/INPUT/EMP/STATEWISE/BA.TXT' INTO TABLE EMP_PARTITION PARTITION (COUNTRY=INDIA,STATE=KA,CITY=BAN);

LOAD DATA LOCAL INPATH='/INPUT/EMP/STATEWISE/HYD.TXT' INTO TABLE EMP_PARTITION PARTITION (COUNTRY=INDIA,STATE=TS,CITY=HYD);

LOAD DATA LOCAL INPATH='/INPUT/EMP/STATEWISE/MUM.TXT' INTO TABLE EMP_PARTITION PARTITION (COUNTRY=INDIA,STATE=MH,CITY=MUM);

LOAD DATA LOCAL INPATH='/INPUT/EMP/STATEWISE/MYSORE.TXT' INTO TABLE EMP_PARTITION PARTITION (COUNTRY=INDIA,STATE=KA,CITY=MYS);

CASE-2: THE DATA DOES HAS PARTITION COLUMN (INSERT).

CREATE TABLE IF NOT EXISTS STOCK_TEMP(symbol string,stock_date string,open double,high double,low double,close double,volume bigint) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/usr/local/hiveinput/staticpartition/NASDAQ_date_bucket_partition.txt' INTO TABLE STOCK_TEMP;

CREATE TABLE IF NOT EXISTS STOCK_PARTITION_DATE(SYMBOL STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT) PARTITIONED BY (STOCK_DATE STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

INSERT OVERWRITE TABLE STOCK_PARTITION_DATE PARTITION (STOCK_DATE='20101105') SELECT SYMBOL,OPEN,HIGH,LOW,CLOSE,VOLUME FROM STOCK_TEMP;

INSERT OVERWRITE TABLE STOCK_PARTITION_DATE PARTITION (STOCK_DATE='20101106') SELECT SYMBOL,OPEN,HIGH,LOW,CLOSE,VOLUME FROM STOCK_TEMP;

INSERT OVERWRITE TABLE STOCK_PARTITION_DATE PARTITION (STOCK_DATE='20101107') SELECT SYMBOL,OPEN,HIGH,LOW,CLOSE,VOLUME FROM STOCK_TEMP;

========== EXAMPLE 2 ==========

CREATE TABLE EMPLOYEE(ID STRING, NAME STRING, SAL BIGINT, COUNTRY STRING)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/usr/local/localinput/partition/empwithcountry.txt' into table employee;

CREATE TABLE EMP_PARTITION_INSERT(ID STRING, NAME STRING, SAL BIGINT) PARTITIONED BY (COUNTRY STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

insert into emp_partition_insert partition (country="IN") select id,name,sal from employee where country="IN";

========== EXAMPLE 3 MULTIPLE PARTITION ==========

CREATE TABLE EMPTEST(ID STRING,NAME STRING, ADDRESS STRING, DOJ BIGINT, PROFFISION STRING, DOB BIGINT,COUNTRY STRING, STATE STRING, CITY STRING)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/user/work/input/hiveinput/EMPLOYEE.txt' INTO TABLE EMPTEST;

CREATE TABLE IF NOT EXISTS EMP_PARTITION(ID STRING, NAME STRING, SAL DOUBLE, DEPT STRING, DOJ STRING) PARTITIONED BY (COUNTRY STRING, STATE STRING, CITY STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

INSERT OVERWRITE TABLE EMP_PARTITION PARTITION (COUNTRY='INDIA',STATE='KA',CITY='BAN') SELECT ID,NAME,ADDRESS, DOJ, PROFFISION, DOB FROM EMPTEST ET WHERE ET.country = 'INDIA' AND ET.state = 'KA' AND ET.CITY='BAN';

INSERT OVERWRITE TABLE EMP_PARTITION PARTITION (COUNTRY='INDIA',STATE='MH',CITY='MUM') SELECT ID,NAME,ADDRESS, DOJ, PROFFISION, DOB FROM EMPTEST ET WHERE ET.country = 'INDIA' AND ET.state = 'MH' AND ET.CITY='MUM';

INSERT OVERWRITE TABLE EMP_PARTITION PARTITION (COUNTRY='INDIA',STATE='TS',CITY='HYD') SELECT ID,NAME,ADDRESS, DOJ, PROFFISION, DOB FROM EMPTEST ET WHERE ET.country = 'INDIA' AND ET.state = 'TS' AND ET.CITY='HYD';

STATIC PARTITION WITH INSERT EXAMPLE 5 (SHOW WE MAY COMMIT MISTAKE BY LOADING/INSERTING WRONG DATA INTO WRONG PARTITION)

CREATE TABLE IF NOT EXISTS STOCK_STAGE(symbol string,stock_date string,open double,high double,low double,close double,volume bigint) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

load data local inpath '/usr/local/hiveinput/staticpartition/NASDAQ_20101105_IN.txt' into table stock_STAGE;

load data local inpath '/usr/local/hiveinput/staticpartition/NASDAQ_20161207_US.csv' into table stock_STAGE;

CREATE TABLE IF NOT EXISTS STOCK_PARTITION_INSERT(symbol string,stock_date string,open double,high double,low double,close double,volume bigint) PARTITIONED BY (country string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

INSERT OVERWRITE TABLE STOCK_PARTITION_INSERT

PARTITION (country = "US")

SELECT * FROM STOCK_STAGE ss

WHERE ss.country = "US";

INSERT OVERWRITE TABLE STOCK_PARTITION_INSERT

PARTITION (country = "IN")

SELECT * FROM STOCK_STAGE ss

WHERE ss.country = "IN";

INSERT OVERWRITE TABLE STOCK_PARTITION_INSERT

PARTITION (country = "UK")

SELECT * FROM STOCK_STAGE ss

WHERE ss.country = "UK";

==you may commit mistake by spacifing wrong partition name=====

INSERT OVERWRITE TABLE STOCK_PARTITION_INSERT

PARTITION (country = "UK")

SELECT * FROM STOCK_STAGE ss

WHERE ss.country = "CA";

LIMITATION OF STATIC PARTITION:

1) WE MAY COMMIT MISTAKES WHILE LOADING OR INSERTING DATA INTO PARTITION TABLE

2) WE NEED TO WRITE SO MANY INSERT OR LOAD STATEMENTS FOR EACH PARTITION.

HIVE PARTITION EMPLOYEE -- DYNAMIC PARTITION

==========

NOTE: NEED TO SET FOLLOWING PROPERTIES..

But by default, Dynamic Partitioning is disabled in Hive to prevent accidental partition creations. To use dynamic partitioning we need to set below properties either in Hive Shell or in hive-site.xml file.

<property>

<name>hive.exec.dynamic.partition</name>

<value>true</value>

<description>Whether or not to allow dynamic partitions in DML/DDL.</description>

</property>

<property> <name>hive.exec.dynamic.partition.mode</name>

<value>nonstrict</value>

<description>

In strict mode, the user must specify at least one static partition in case the user accidentally overwrites all partitions.

In nonstrict mode all partitions are allowed to be dynamic.

</description>

</property>

<property> <name>hive.exec.max.dynamic.partitions</name>

<value>1000</value>

<description>Maximum number of dynamic partitions allowed to be created in total.</description>

</property>

<property>

<name>hive.exec.max.dynamic.partitions.pernode</name>

<value>1000</value>

<description>Maximum number of dynamic partitions allowed to be created in each mapper</description>

</property>

We can set these through hive shell with below commands,

set hive.exec.dynamic.partition=true;

set hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.max.dynamic.partitions=1000;

set hive.exec.max.dynamic.partitions.pernode=1000;

==========Sample data==========

NASDAQ,JDAS,2010-01-29,26.91,27.53,26.02,26.21,883100,26.21

NASDAQ,JDAS,2010-01-28,27.86,27.97,26.84,26.88,1272600,26.88

NASDAQ,JDAS,2010-01-27,27.48,27.93,27.20,27.68,560100,27.68

NASDAQ,JDAS,2010-02-08,25.41,26.59,25.15,26.46,488900,26.46

NASDAQ,JDAS,2010-02-05,25.42,25.84,24.94,25.49,1121700,25.49

NASDAQ,JDAS,2010-02-04,26.53,26.61,25.46,25.46,574900,25.46

NASDAQ,JDAS,2009-12-31,25.97,26.13,25.47,25.47,283600,25.47

NASDAQ,JDAS,2009-12-30,25.74,26.25,25.61,26.05,236300,26.05

NASDAQ,JDAS,2009-12-29,25.98,25.98,25.52,25.76,238600,25.76

NASDAQ,JDAS,2009-11-30,23.39,23.65,22.78,23.48,522000,23.48

NASDAQ,JDAS,2009-11-27,23.12,23.71,23.10,23.54,144900,23.54

NASDAQ,JDAS,2009-11-25,23.96,24.00,23.59,23.82,220400,23.82

NASDAQ,JOEZ,2010-01-29,1.68,1.69,1.60,1.60,158900,1.60

NASDAQ,JOEZ,2010-01-28,1.64,1.70,1.61,1.62,250700,1.62

NASDAQ,JOEZ,2010-01-27,1.73,1.76,1.63,1.64,329200,1.64

NASDAQ,JOEZ,2010-01-26,1.70,1.76,1.66,1.70,509100,1.70

NASDAQ,JOEZ,2010-01-25,1.64,1.68,1.60,1.68,169600,1.68

NASDAQ,JOEZ,2010-02-08,1.80,2.04,1.76,1.93,1712200,1.93

NASDAQ,JOEZ,2010-02-05,1.84,1.88,1.70,1.80,1044700,1.80

NASDAQ,JOEZ,2010-02-04,1.96,1.97,1.74,1.88,3758600,1.88

NASDAQ,JOEZ,2010-02-03,1.73,1.79,1.68,1.72,1211700,1.72

NASDAQ,JOEZ,2010-02-02,1.59,1.72,1.51,1.70,909400,1.70

NASDAQ,JOEZ,2009-07-15,1.00,1.05,0.75,0.81,1215200,0.81

NASDAQ,JOEZ,2009-07-14,0.80,0.95,0.80,0.93,580000,0.93

```
CREATE TABLE IF NOT EXISTS STOCK_STAGE(EXCHNGE STRING,SYMBOL STRING,DATE STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT,ADJ_CLOSE DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/user/work/input/StockData.txt' into table STOCK_STAGE;

CREATE TABLE IF NOT EXISTS STOCK_PARTITION(EXCHNGE STRING,SYMBOL STRING,DATE STRING,OPEN DOUBLE,HIGH DOUBLE,LOW DOUBLE,CLOSE DOUBLE,VOLUME BIGINT,ADJ_CLOSE DOUBLE) PARTITIONED BY (DT STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

INSERT OVERWRITE TABLE STOCK_PARTITION PARTITION (DT)   SELECT EXCHNGE,SYMBOL,DATE,OPEN,HIGH,LOW,CLOSE,VOLUME,ADJ_CLOSE, SS.DATE AS DT FROM STOCK_STAGE SS WHERE DATE IS NOT NULL;

==========example 2==========

DROP TABLE IF EXISTS partitioned_user;

CREATE TEMPORARY TABLE temp_user(

firstname STRING, lastname STRING,

address STRING, country STRING,

city STRING, state STRING, post STRING,

phone1 STRING, phone2 STRING,

email STRING, web STRING

)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;


LOAD DATA LOCAL INPATH '/home/veera/UserRecords.txt' INTO TABLE temp_user;

SELECT firstname, phone1, city FROM temp_user WHERE country='US' AND state='CA'  ORDER BY city limit 10;

CREATE TABLE partitioned_user(

firstname STRING, lastname STRING,  address STRING, city STRING, post STRING, phone1 STRING,  phone2 STRING, email STRING, web STRING ) PARTITIONED BY (country STRING, state STRING) STORED AS SEQUENCEFILE;


INSERT INTO TABLE partitioned_user PARTITION (country, state)

SELECT firstname ,lastname ,address ,city ,post ,phone1 ,phone2 ,email ,web ,country ,state FROM temp_user;


SELECT firstname, phone1, city FROM partitioned_user WHERE country='US' AND state='CA' ORDER BY city LIMIT 5;

==========
```

Show partitions:

```
hive> SHOW PARTITIONS partitioned_user;

hive> SHOW PARTITIONS partitioned_user PARTITION(country='US');
```

==========

```
hive> DESCRIBE FORMATTED partitioned_user PARTITION(country='US', state='CA');
OK
# col_name      data_type       comment
firstname       STRING
lastname        STRING
address string
city    STRING
post    string
phone1 STRING
phone2 string
email   string
web     string
# Partition Information
# col_name      data_type       comment
country STRING
state   STRING
# Detailed Partition Information
Partition Value: [US, CA]
Database:       default
Table:  partitioned_user
CreateTime:     Tue Dec 09 22:34:30 IST 2014
LastAccessTime:         UNKNOWN
Protect Mode:   None
Location:       hdfs://localhost:9000/user/hive/warehouse/partitioned_user/country
Partition Parameters:
COLUMN_STATS_ACCURATE   true
```

numFiles 1

numRows 72

rawDataSize 9358

totalSize 10527

transient_lastDdlTime 1418144688

==========

ALTER TABLE partitioned_user ADD IF NOT EXISTS

PARTITION (country = 'US', state = 'XY') LOCATION '/hdfs/external/file/path1'

PARTITION (country = 'CA', state = 'YZ') LOCATION '/hdfs/external/file/path2'

PARTITION (country = 'UK', state = 'ZX') LOCATION '/hdfs/external/file/path2'

CHANGING PARTITIONS:

We can change a partition location with commands like below. This command does not move the data from the old location and does not delete the old data but the reference to old data file will be lost.

ALTER TABLE partitioned_user PARTITION (country='US', state='CA') SET LOCATION '/hdfs/partition/newpath';

==========

Drop Partitions:

We can drop partitions of a table with DROP IF EXISTS PARTITION clause as shown below.

ALTER TABLE partitioned_user DROP IF EXISTS PARTITION(country='US', state='CA');

==========

The ARCHIVE PARTITION clause captures the partition files into a Hadoop archive (HAR) file.This only reduces the number of files in the filesystem, reducing the load on the NameNode, but doesn't provide any space savings.

ALTER TABLE log_messages ARCHIVE PARTITION(country='US',state='XZ');

we can un archive these with UNARCHIVE PARTITION clause.

The following statements prevent the partition from being dropped and queried.

ALTER TABLE partitioned_user PARTITION(country='US',state='XY') ENABLE NO_DROP;

ALTER TABLE partitioned_user PARTITION(country='US',state='XY') ENABLE OFFLINE;

==========

hive -e "select * from categories_location LIMIT 5"

hive -f queries.sql

-- Adding partition manually

alter table orders_part_avro add partition (order_month='2014-01');

-- Inserting data to a partition

insert into table orders_part_avro partition (order_month='2014-01')

select * from orders where from_unixtime(cast(substr(order_date, 1, 10) as int)) like '2014-01%';

****

BUCKETING

****

CREATE TABLE IF NOT EXISTS STOCK(symbol string,stock_date string,open double,high double,low double,close double,volume bigint) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

==========BUCKETING WITHOUT PARTITION==========

CREATE TABLE IF NOT EXISTS STOCK_BUCKET(symbol string,stock_date string,open double,high double,low double,close double,volume bigint) CLUSTERED BY (symbol) INTO 32 BUCKETS STORED AS TEXTFILE;

INSERT OVERWRITE TABLE STOCK_BUCKET SELECT * FROM STOCK;

==========BUCKETING WITH PARTITION==========

set hive.exec.dynamic.partition.mode=nonstrict

DROP TABLE STOCK_DATE;

CREATE TABLE IF NOT EXISTS STOCK_DATE(symbol string,stock_date string,open double,high double,low double,close double,volume bigint) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

load data local inpath '/usr/local/hiveinput/staticpartition/NASDAQ_date_bucket_partition.txt' into table stock_date;

DROP TABLE STOCK_BUCKET_PARTITION;

CREATE TABLE IF NOT EXISTS STOCK_BUCKET_PARTITION(symbol string,open double,high double,low double,close double,volume bigint) PARTITIONED BY (STOCK_DATE STRING) CLUSTERED BY (symbol) INTO 32 BUCKETS ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

INSERT OVERWRITE TABLE STOCK_BUCKET_PARTITION PARTITION (STOCK_DATE) SELECT SYMBOL,OPEN,HIGH,LOW,CLOSE,VOLUME,STOCK_DATE FROM STOCK_DATE;

CREATE TABLE bucketed_tbl3 (id INT, name STRING) CLUSTERED BY (id) INTO 4 BUCKETS;

==========Buckets Table Sampling==========

set hive.enforce.bucketing = true;

set map.reduce.tasks = 4;

Bucket without partition:

i) CREATE TABLE bucketed_tbl3 (id INT, name STRING) CLUSTERED BY (id) INTO 4 BUCKETS;

ii)CREATE TABLE users2 (id INT, name STRING)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

0,veera1

2,veera2

3,veera3

4,veera4

iii) load data local inpath '/home/user/work/input/HiveBucketInput.txt' into table users2;

iv) select * from users2;

v)INSERT OVERWRITE TABLE bucketed_tbl3 SELECT * FROM users2;

n = total no of buckets = 4

TABLESAMPLE(BUCKET 1 OUT OF 4)

n/y :- 4/4 = 1 group which contains (1,2,3,4)' so it will pick 1st bucket out of group of 4.

TABLESAMPLE(BUCKET 2 OUT OF 8)

4/8 = ½ (half group, 1,2,3,4….8 we have only 4 buckets, so 5,6,7,8 are not buckets) so it will pick 2nd bucket

How Sampling Works:

When you create the table and bucket it using the clustered by clause into 32 buckets (as an example), hive buckets your data into 32 buckets using deterministic hash functions. Then when you use TABLESAMPLE(BUCKET x OUT OF y), hive divides your buckets into groups of y buckets and then picks the xth bucket of each group. For example:

 If you use TABLESAMPLE(BUCKET 6 OUT OF 8), hive would divide your 32 buckets into groups of 8 buckets resulting in 4 groups of 8 buckets and then picks the 6th bucket of each group, hence picking the buckets 6, 14, 22, 30.

 If you use TABLESAMPLE(BUCKET 23 OUT OF 32), hive would divide your 32 buckets into groups of 32, resulting in only 1 group of 32 buckets, and then picks the 23rd bucket as your result.

 If you use TABLESAMPLE(BUCKET 3 OUT OF 64), hive would divide your 32 buckets into groups of 64 buckets, resulting in 1 group of 64 “half-bucket”s and then picks the half-bucket that corresponds to the 3rd full-bucket.

vi) SELECT * FROM bucketed_tbl3 TABLESAMPLE(BUCKET 1 OUT OF 4 ON id);

Bucket with Partitions:

7) create table sample_buckets1(key int, value string) partitioned by (dt STRING) CLUSTERED BY(value) SORTED BY(key) INTO 5 BUCKETS ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

8) load data local inpath '/home/prime23/kumar/Work/input/k1.log' into table sample_buckets1 PARTITION (dt='87889');

9) select * from sample_buckets1 where dt='87889';

Note: Set the no of reducers = no of buckets

set map.reduce.tasks = 25

set hive.enforce.bucketing = true;

NOTE: In partitioned tables when we issue a query only the required partitions are scanned, no need to specify any hints in your hive query. But for bucketed tables it is not the case, you need to hint your hive query if you want to scan some particular buckets else the whole set of files would be scanned. We hint the buckets using TABLESAMPLE clause in our hive query. For example in our example if we want to choose only the data from BUCKET 2

9) SELECT * FROM sample_buckets1 TABLESAMPLE(BUCKET 2 OUT OF 5 ON key);

https://cwiki.apache.org/confluence/display/Hive/Tutorial#Tutorial-Sampling

---

MULTITALE INSERT

****

CREATE TABLE EMPLOYEETEST(ID INT,NAME STRING,SAL DOUBLE,DEPTID STRING,DEPTNAME STRING,LOCATION STRING,COUNTRY STRING,DEDUCTION DOUBLE,PF DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/user/work/input/hiveinput/employeewithdept.txt' INTO TABLE EMPLOYEETEST;

CREATE TABLE EMPDETAILS(ID INT,NAME STRING,SAL DOUBLE,DEPTID STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

CREATE TABLE DEPTDETAILS(DEPTID STRING,DEPTNAME STRING,LOCATION STRING,COUNTRY STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

CREATE TABLE EMPDEDUCTIONS1(ID INT,NAME STRING,DEDUCTION DOUBLE,PF DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

FROM EMPLOYEETEST INSERT OVERWRITE TABLE EMPDETAILS SELECT ID,NAME,SAL,DEPTID INSERT OVERWRITE TABLE DEPTDETAILS SELECT DEPTID,DEPTNAME,LOCATION,COUNTRY INSERT OVERWRITE TABLE EMPDEDUCTIONS1 SELECT ID,NAME,DEDUCTION,PF;

SELECT * FROM EMPLOYEETEST;

SELECT * FROM EMPDETAILS;

SELECT * FROM DEPTDETAILS;

SELECT * FROM EMPDEDUCTIONS1;

JOIN

****

CREATE TABLE EMP(ID STRING, NAME STRING, SAL BIGINT)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

load data local inpath '/usr/local/localinput/partition/emp_in.txt' into table emp;

CREATE TABLE EMAILID(NAME STRING, EMAIL STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/usr/local/localinput/partition/emailid.txt' into table emailid;

SELECT EMP.ID,EMP.NAME,EMP.SAL,EMAILID.EMAIL FROM EMP JOIN EMAILID ON EMP.NAME = EMAILID.NAME;

==========

Total MapReduce CPU Time Spent: 1 seconds 660 msec

OK

1 ravi 1000 veeraravi2110@gmail.com

2 veera 2000 veeraravi@gmail.com

4 kumar 3456 kumar.s@gmail.com

3 singiri 34567 singiri@gmail.com

Time taken: 27.8 seconds, Fetched: 4 row(s)

SELECT A.KEY AS SELECTEDKEY,A.VALUE AS SELECTEDVALUE FROM SAMPLE1 AS A JOIN SAMPLE3 AS B ON A.KEY = B.KEY WHERE A.KEY >=100;

CREATE TABLE EMPDETAILS(ID INT,NAME STRING,SAL DOUBLE,DEPTID STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

CREATE TABLE DEPTDETAILS(DEPTID STRING,DEPTNAME STRING,LOCATION STRING,COUNTRY STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

SELECT * FROM EMPDETAILS;

101 ravi 10000.0 d101

102 veera 13000.0 d101

103 saurabh 10000.0 d102

104 manju 10000.0 d101

105 muni 70000.0 d103

106 shruthi 20000.0 d102

107 kumar 50000.0 d104

Time taken: 1.671 seconds, Fetched: 7 row(s)

hive> select * from deptdetails;

OK

d101 bigdata bangalore india

d101 bigdata bangalore india

d102 hadoop delhi india

d101 bigdata bangalore india

d103 JAVA/BIGDATA bangalore india

d102 hadoop bangalore india

d104 java bangalore india

Time taken: 0.196 seconds, Fetched: 7 row(s)

SELECT A.ID,A.NAME,B.DEPTNAME,B.COUNTRY FROM EMPDETAILS A JOIN DEPTDETAILS B ON A.DEPTID = B.DEPTID;

===Left out Join

SELECT A.ID,A.NAME,B.DEPTNAME,B.COUNTRY FROM EMPDETAILS A LEFT OUTER JOIN DEPTDETAILS B ON A.DEPTID = B.DEPTID;

====Right outer Join

SELECT A.ID,A.NAME,B.DEPTNAME,B.COUNTRY FROM EMPDETAILS A RIGHT OUTER JOIN DEPTDETAILS B ON A.DEPTID = B.DEPTID;

==========FULL outer Join

SELECT A.ID,A.NAME,B.DEPTNAME,B.COUNTRY FROM EMPDETAILS A FULL OUTER JOIN DEPTDETAILS B ON A.DEPTID = B.DEPTID;

==========example 2========

CREATE TABLE CUST(ID INT,NAME STRING,PID STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

CREATE TABLE PROD(DPID STRING,PNAME STRING,PDCRIPTION STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/HOME/USER/WORK/INPUT/HIVEINPUT/CUSTPRODID.TXT' INTO TABLE CUST;

LOAD DATA LOCAL INPATH '/HOME/USER/WORK/INPUT/HIVEINPUT/PRODCTDETAILS.TXT' INTO TABLE PROD;

CREATE TABLE EMPDEDUCTIONS1(ID INT,NAME STRING,DEDUCTION DOUBLE,PF DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\T' STORED AS TEXTFILE;

==========

UDF

==========

UDF's:

3) create table udfTable3(id int, unixtime bigint)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

4) load data local inpath '/home/user/work/input/udfInput10.txt' into table udfTable3;

5) select * from udfTable3;

6) show functions;

7) add jar /home/prime23/Work/hiveUDF00.jar;

8)register the udf function

hive>create temporary function unixtimetodate as 'veera.hadoop.hive.udf.examples.UnixSystemTimeToDate';

9) show functions;

10) use it as below

Example:

create table

hive>select id, unixtime from udfTable3;

12 979959582

17 652781043

67 NULL

Then use function 'unixtimetodate'

hive>select id, unixtimetodate(unixtime) from udfTable3;

12 19/11/97 10:43 PM

==========

UDF: Example 1: CustomGenericUDFHive-NVL2

covers a simple Hive genericUDF in Java, that mimics NVL2 functionality in Oracle.

NVL2 is used to handle nulls and conditionally substitute values.

Included:

1. Input data
2. Expected results
3. UDF code in java
4. Hive query to demo the UDF
5. Output

Note: The dataset is very small - as the purpose of this gist is instructional. :)

About the NVL2 functionality in this demo:

Return type: String

Parameters: Three comma separated strings, we will refer to as:

expr1, expr2, expr3

Purpose: If expr1 is null, NVL2 returns expr3, otherwise, expr3

****

Input data

****

1. Execute locally on the node you are running Hive client from

---

Create input file/data to use for the demo.

Since this gist is merely for instructional purpose, the dataset is small.

cd ~

mkdir hiveProject

cd hiveProject

vi Departments_UDFTest

Paste this..ensuring the fields are delimited by tabs and record with new line.

d001 Marketing

d002 Finance

d003 Human Resources

d004 Production

d005 Development

d006 Quality Management

d007 Sales

d008

d009 Customer Service

2. Hadoop commands

---

hadoop fs -mkdir hiveProject

hadoop fs -put hiveProject/Departments_UDFTest hiveProject

****

Setting up the Hive table

****

In hive shell....

a) Create table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS departments_UDFTest
(
deptNo String,
deptName String
)
Row format delimited
fields terminated by '\t'
LOCATION '/user/akhanolk/hiveProject';
```

b) Quick test:

```
Select * from departments_UDFTest;
d001    Marketing
d002    Finance
d003    Human Resources
d004    Production
d005    Development
d006    Quality Management
d007    Sales
d008    NULL
d009    Customer Service
```

========== JAVA CODE:

```
package khanolkar.HiveUDFs;
import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.exec.UDFArgumentLengthException;
import org.apache.hadoop.hive.ql.exec.UDFArgumentTypeException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.UDFType;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDF;
```

```
import org.apache.hadoop.hive.ql.udf.generic.GenericUDFUtils;

import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;

@UDFType(deterministic = true)

@Description(name = "NVL2", value = "NVL2(expr1,expr2,expr3) returns expr3, if expr1 is null, otherwise returns
expr2;", extended = "NVL2(expr1,expr2,expr3) returns expr3, if expr1 is null, otherwise retruns expr2")

public class NVL2GenericUDF extends GenericUDF {

        private GenericUDFUtils.ReturnObjectInspectorResolver returnOIResolver;

        private ObjectInspector[] argumentOIs;

        @Override

        public ObjectInspector initialize(ObjectInspector[] arguments)

                        throws UDFArgumentException {

                argumentOIs = arguments;

                // First check - do we have the right number of arguments?

                if (arguments.length != 3) {

                        throw new UDFArgumentLengthException(

                                        "The operator 'NVL2'  accepts 3 arguments.");

                }

                // Second check - throw exception if any complex types have been passed

                // as parameters

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

                                || arguments[1].getCategory() != ObjectInspector.Category.PRIMITIVE

                                || arguments[2].getCategory() != ObjectInspector.Category.PRIMITIVE)

                        throw new UDFArgumentTypeException(0,

                                        "Only primitive type arguments are accepted");

                // Third check - throw exception if the data types across parameters are

                // different

                if (!(returnOIResolver.update(arguments[0]) && returnOIResolver

                                .update(arguments[1]))

                                || !(returnOIResolver.update(arguments[1]) && returnOIResolver

                                                .update(arguments[2]))) {

                        throw new UDFArgumentTypeException(2,
```

```
                                "The arguments of function NLV2 should have the same type, "
                                        + "but they are different: \""
                                        + arguments[0].getTypeName() + "\" and \""
                                        + arguments[1].getTypeName() + "\" and \""
                                        + arguments[2].getTypeName() + "\"");
        }
        returnOIResolver = new GenericUDFUtils.ReturnObjectInspectorResolver(
                        true);
        return returnOIResolver.get();
}
@Override
public Object evaluate(DeferredObject[] arguments) throws HiveException {
        // The NVL2 functionality
        Object retVal = returnOIResolver.convertIfNecessary(arguments[0].get(),
                        argumentOIs[0]);
        if (retVal == null) {
                retVal = returnOIResolver.convertIfNecessary(arguments[2].get(),
                                argumentOIs[2]);
        } else {
                retVal = returnOIResolver.convertIfNecessary(arguments[1].get(),
                                argumentOIs[1]);
        }
        return retVal;
}
@Override
public String getDisplayString(String[] children) {
        StringBuilder sb = new StringBuilder();
        sb.append("if ");
        sb.append(children[0]);
        sb.append(" is null ");
```

```
                sb.append("returns");
                sb.append(children[2]);
                sb.append("else ");
                sb.append("returns");
                sb.append(children[1]);
                return sb.toString();
        }
}
```

==========

Concepts

There are three methods-

1. initialize() - called once, at first. The goal of this method is to determine the return type from the arguments. The user can also throw an Exception to signal that bad types are being sent to the method. The returnOIResolver is a built-in class that determines the return type by finding the type of non-null variables and using that type. The ObjectInspector is used to the transform raw records into objects that Hive can access. The initialize() method is passed an ObjectInspector for each argument

2. evaluate() - where the logic for the function should be written.

The evaluate method has access to the values passed to the method stored in an array of DeferredObject values. The returnOIResolver created in the initialize method is used to get values from the DeferredObjects.

3. getDisplayString() - The final method to override is getDisplayString(), is used inside the Hadoop tasks to display debugging information when the function is being used.

Annotations:

@UDFType(deterministic = true) annotation: Indicates that the UDF returns the same value any time its called

@Description(...) annotation: Includes information that is displayed when you do a describe on the UDF

****

Testing the UDF

****

1) Add jar

hive> add jar hiveProject/jars/NVL2GenericUDF.jar;

2) Create alias for the function

hive> CREATE TEMPORARY FUNCTION NVL2

AS 'khanolkar.HiveUDFs.NVL2GenericUDF';

3) Test the description provided

hive> DESCRIBE FUNCTION NVL2;

NVL2(expr1,expr2,expr3) returns expr3, if expr1 is null, otherwise returns expr2;

4) Test if there are checks in place for number of parameters

hive> select deptNo,NVL2(deptName,deptName) from departments_UDFTest;

FAILED: SemanticException [Error 10015]: Line 1:14 Arguments length mismatch 'deptName': The operator 'NVL2' accepts 3 arguments.

hive> select deptNo,NVL2(deptName,deptName,123,1) from departments_UDFTest;

FAILED: SemanticException [Error 10015]: Line 1:14 Arguments length mismatch '1': The operator 'NVL2' accepts 3 arguments.

5) Results

hive> select deptNo,NVL2(deptName,deptName,'Procrastrination') from departments_UDFTest;

OK

d001 Marketing

d002 Finance

d003 Human Resources

d004 Production

d005 Development

d006 Quality Management

d007 Sales

d008 Procrastrination

d009 Customer Service

UDF: Example 2: CustomGenericUDFHive-NVL2

covers a simple Hive eval UDF in Java, that mimics NVL2 functionality in Oracle.

NVL2 is used to handle nulls and conditionally substitute values.

Included:

1. Input data
2. Expected results
3. UDF code in java
4. Hive query to demo the UDF
5. Output

Note: The dataset is very small - as the purpose of this gist is instructional. :)

About the NVL2 functionality in this demo:

Return type: String

Parameters: Three comma separated strings, we will refer to as:

expr1, expr2, expr3

Purpose: If expr1 is null, NVL2 returns expr3, otherwise, expr3

****

Input data

****

1. Execute locally on the node you are running Hive client from

---

Create input file/data to use for the demo.

Since this gist is merely for instructional purpose, the dataset is small.

cd ~

mkdir hiveProject

cd hiveProject

vi Departments_UDFTest

Paste this..ensuring the fields are delimited by tabs and record with new line.

d001 Marketing

d002 Finance

d003 Human Resources

d004 Production

d005 Development

d006 Quality Management

d007 Sales

d008

d009 Customer Service

2. Hadoop commands

---

hadoop fs -mkdir hiveProject

hadoop fs -put hiveProject/Departments_UDFTest hiveProject

****

Setting up the Hive table

****

In hive shell....

a) Create table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS departments_UDFTest
(
deptNo String,
deptName String
)
Row format delimited
fields terminated by '\t'
LOCATION '/user/akhanolk/hiveProject';
```

b) Quick test:

Select * from departments_UDFTest;

d001 Marketing

d002 Finance

```
d003    Human Resources
d004    Production
d005    Development
d006    Quality Management
d007    Sales
d008    NULL
d009    Customer Service
//-----------------------------------------------------------------
// Filename: NVL2.java
//-----------------------------------------------------------------
package khanolkar.HiveUDFs;
import java.io.IOException;
import org.apache.hadoop.hive.ql.exec.UDF;
public class NVL2 extends UDF {
      String expr1, expr2, expr3;
          public NVL2() {  }
          public String evaluate(String pExpr1, String pExpr2, String pExpr3)
                              throws IOException {
                    try {     expr1 = (String) pExpr1;
                              expr2 = (String) pExpr2;
                              expr3 = (String) pExpr3;
                              return (expr1 != null ? expr2 : expr3);
                    } catch (Exception e) {
                              // Cause task failure
                              throw new IOException("Error with Hive UDF, NVL2!", e);
                    }
          }
}
```

****

Expected results

****

Query:

select deptNo,NVL2(deptName,deptName,'Procrastrination') from departments_UDFTest;

The null in the department name for department d008, should be returned as "Procrastrination".

For the rest of the records, the query should return the data in Hive, as is.

****

Testing the UDF

****

hive> add jar hiveProject/jars/NVL2.jar;

hive> CREATE TEMPORARY FUNCTION NVL2

AS 'khanolkar.HiveUDFs.NVL2';

hive> select deptNo,NVL2(deptName,deptName) from departments_UDFTest;

FAILED: SemanticException [Error 10014]: Line 1:14 Wrong arguments 'deptName': No matching method for class khanolkar.HiveUDFs.NVL2 with (string, string). Possible choices: _FUNC_(string, string, string)

hive> select deptNo,NVL2(deptName,deptName,'Procrastrination') from departments_UDFTest;

OK

d001 Marketing

d002 Finance

d003 Human Resources

d004 Production

d005 Development

d006 Quality Management

d007 Sales

d008 Procrastrination

d009 Customer Service

****

SERDE

****

Serde's:

1) add jar /home/user/work/apache-hive-0.13.1-bin/hive-contrib/hive-json-serde-0.2.jar

2) Create some Input JSON data

{"field1":"data1","field2":100,"field3":"more data1","field4":123.001}

{"field1":"data2","field2":200,"field3":"more data2","field4":123.002}

{"field1":"data3","field2":300,"field3":"more data3","field4":123.003}

{"field1":"data4","field2":400,"field3":"more data4","field4":123.004}

3)Create a table:

```
CREATE TABLE IF NOT EXISTS my_table2 (
        field1 string, field2 int, field3 string, field4 double
    )    ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.JsonSerde';
```

4) LOAD DATA LOCAL INPATH '/home/user/work/input/sample_serde_input.json' INTO TABLE my_table2;

5)select * from my_table2;

insert overwrite local directory '/home/prime23/Work/input/hiveResults' selectrime23/Work/input/hiveResults' select field1 from my_table2;

Example:

Sample data

May 3 11:52:54 cdh-dn03 init: tty (/dev/tty6) main process (1208) killed by TERM signal

May 3 11:53:31 cdh-dn03 kernel: registered taskstats version 1

May 3 11:53:31 cdh-dn03 kernel: sr0: scsi3-mmc drive: 32x/32x xa/form2 tray

May 3 11:53:31 cdh-dn03 kernel: piix4_smbus 0000:00:07.0: SMBus base address uninitialized - upgrade BIOS or use force_addr=0xaddr

May 3 11:53:31 cdh-dn03 kernel: nf_conntrack version 0.5.0 (7972 buckets, 31888 max)

May 3 11:53:57 cdh-dn03 kernel: hrtimer: interrupt took 11250457 ns

May 3 11:53:59 cdh-dn03 ntpd_initres[1705]: host name not found: 0.rhel.pool.ntp.org

Structure

Month = May

Day = 3

Time = 11:52:54

Node = cdh-dn03

Process = init:

Log msg = tty (/dev/tty6) main process (1208) killed by TERM signal

a) Load the data

$ hadoop fs -mkdir LogParserSampleHive

$ hadoop fs -mkdir LogParserSampleHive/logs

$ hadoop fs -put LogParserSampleHive/logs/* LogParserSampleHive/logs/

$ hadoop fs -ls -R LogParserSampleHive/ | awk {'print $8'}

Hive commands

---

a) Create external table:

```
hive> CREATE EXTERNAL TABLE LogParserSample(
month_name STRING,
day STRING,
time STRING,
host STRING,
event STRING,
log STRING)
PARTITIONED BY(year int, month int)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
 "input.regex" = "(\\w+)\\s+(\\d+)\\s+(\\d+:\\d+:\\d+)\\s+(\\w+\\W*\\w*)\\s+(.*?\\:)\\s+(.*$)"
) stored as textfile;
```

b) Create partitions and load data:

Note: Replace '/user/airawat' with '/user/<your userID>'

```
hive> Alter table LogParserSample Add IF NOT EXISTS partition(year=2013, month=04)
    location '/user/airawat/LogParserSampleHive/logs/airawat-syslog/2013/04/';
hive> Alter table LogParserSample Add IF NOT EXISTS partition(year=2013, month=05)
    location '/user/airawat/LogParserSampleHive/logs/airawat-syslog/2013/05/';
```

Hive query

```
hive> set hive.cli.print.header=true;
hive> add jar hadoop-lib/hive-contrib-0.10.0-cdh4.2.0.jar; --I need this as my environment is not properly configured
hive> select Year,Month,Day,Event,Count(*) Occurrence from LogParserSample group by year,month,day,event order by event desc,year,month,day;
```

Query output

---

year	month	day	event	occurrence
2013	05	7	udevd[361]:	1
2013	04	23	sudo:	1
2013	05	3	sudo:	1
2013	05	3	ntpd_initres[1705]:	144
2013	05	4	ntpd_initres[1705]:	261
2013	05	5	ntpd_initres[1705]:	264
2013	05	6	ntpd_initres[1705]:	123
2013	05	3	kernel:	5
2013	05	6	kernel:	1
2013	05	7	kernel:	52
2013	05	3	init:	5
2013	05	7	init:	18

****

Find Value length of column in <K,V>

****

```
add jar /home/user/Desktop/EXAMPLEJARS/hivejars/valueLength.jar;

create temporary function getColumnValue as 'com.hive.customudfs.ColunValueLength';

create table findlength(key int,value string)row format delimited fields terminated by ',' stored as textfile;

load data local inpath '/home/user/work/input/hiveinput/simpletext.txt' into table findlength;

select key,getColumnValue(value) from findlength;
```

****

Employee Max salary by dept

****

```
CREATE TABLE IF NOT EXISTS EMP_SAL(id int, name String, dob String, salary Double, department String, address string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/user/work/input/hiveinput/employeedata.txt' into table EMP_SAL;

SELECT DEPARTMENT,MAX(SALARY) FROM EMP_SAL GROUP BY DEPARTMENT;

WITH cteRowNum AS (
   SELECT id, name, dob,salary,department,address
        DENSE_RANK() OVER(PARTITION BY DeptID ORDER BY Salary DESC) AS RowNum
      FROM emp_sal
)SELECT id, name, dob,salary,department,address
   FROM cteRowNum
   WHERE RowNum = 1;
```

****

Replace characters

****

```
select regexp_replace(start_date, '-', '') from test;

or

SET REGEX="(\\d{2})-(\\d{2})-(\\d{4})"; SELECT  CONCAT(    regexp_extract(dob, ${hiveconf:REGEX}, 1), regexp_extract(dob, ${hiveconf:REGEX}, 2),    regexp_extract(dob, ${hiveconf:REGEX}, 3)  )

FROM emp_sal;
```

****

Parameter

****

1) When doing shell scripts, executing lines have to be wrapped with `. so i did

temp.sh

```
temp=`date --date='yesterday' +%y%m%d`
hive -f testing.hql -hiveconf var=$temp
```

and it works like a charm

2) in the query, the parameter must be in double quotes.

hive.hdl

```
select jobid from temp_table where dt >= "${hiveconf:var}";
```

Hope this question can help others who had this issue.

****

RESULT TO CSV file

****

```
hive -e "select a.key as selectedKey,a.value as selectedValue from sample1 as a JOIN sample3 as b ON a.key = b.key where a.key >=200" > /home/user/work/veerahive.tsv
```

****

INDEXING

****

```
CREATE TABLE IF NOT EXISTS STOCK(symbol string,stock_date string,open double,high double,low double,close double,volume bigint) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

CREATE INDEX STOCK_INDEX ON TABLE STOCK(OPEN) AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;

ALTER INDEX STOCK_INDEX on STOCK REBUILD;

DROP INDEX IF EXISTS STOCK_INDEX ON STOCK;
```

==========

ACID PROPERTIES

==========

Configuration

Minimally, these configuration parameters must be set appropriately to turn on transaction support in Hive:

======Client Side=======

hive.support.concurrency – true

hive.enforce.bucketing – true (Not required as of Hive 2.0)

hive.exec.dynamic.partition.mode – nonstrict

hive.txn.manager – org.apache.hadoop.hive.ql.lockmgr.DbTxnManager

=======Server Side (Metastore)==========

hive.compactor.initiator.on – true (See table below for more details)

hive.compactor.worker.threads – a positive number on at least one instance of the Thrift metastore service

==========Configuration Values to Set for INSERT, UPDATE, DELETE==========

In addition to the new parameters listed above, some existing parameters need to be set to support INSERT ... VALUES, UPDATE, and DELETE.

Configuration key	Must be set to
hive.support.concurrency	true (default is false)
hive.enforce.bucketing	true (default is false) (Not required as of Hive 2.0)
hive.exec.dynamic.partition.mode	nonstrict (default is strict)

Configuration Values to Set for Compaction

If the data in your system is not owned by the Hive user (i.e., the user that the Hive metastore runs as), then Hive will need permission to run as the user who owns the data in order to perform compactions. If you have already set up HiveServer2 to impersonate users, then the only additional work to do is assure that Hive has the right to impersonate users from the host running the Hive metastore. This is done by adding the hostname to hadoop.proxyuser.hive.hosts in Hadoop's core-site.xml file. If you have not already done this, then you will need to configure Hive to act as a proxy user. This requires you to set up keytabs for the user running the Hive metastore and add hadoop.proxyuser.hive.hosts and hadoop.proxyuser.hive.groups to Hadoop's core-site.xml file. See the Hadoop documentation on secure mode for your version of Hadoop

refer :https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions

hive>set hive.txn.manager = org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;

hive>set hive.compactor.initiator.on = true;

hive>set hive.compactor.worker.threads = a positive number on at least one instance of the Thrift metastore service;

hive>set hive.support.concurrency = true;

hive>set hive.enforce.bucketing = true;

hive>set hive.exec.dynamic.partition.mode = nonstrict;

CREATE TABLE COLLEGE(CLG_ID INT,CLG_NAME STRING,CLG_LOC STRING) CLUSTERED BY (CLG_ID) INTO 5 BUCKETS STORED AS ORC TBLPROPERTIES('TRANSACTIONAL'='TRUE');

****

insert stmt IT WILL WORK IN HIVE 0.14

****

CREATE TABLE students (name VARCHAR(64), age INT, gpa DECIMAL(3, 2))

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;

INSERT INTO TABLE students VALUES ('fred flintstone', 35, 1.28), ('barney rubble', 32, 2.32),('barney rubble', 32, 2.32);

CREATE TABLE college(clg_id int,clg_name string,clg_loc string) clustered by (clg_id) into 5 buckets stored as orc TBLPROPERTIES('transactional'='true');

CREATE TABLE pageviews (userid VARCHAR(64), link STRING, from STRING)

PARTITIONED BY (datestamp STRING) CLUSTERED BY (userid) INTO 256 BUCKETS STORED AS ORC;

INSERT INTO TABLE pageviews PARTITION (datestamp = '2014-09-23')

VALUES ('jsmith', 'mail.com', 'sports.com'), ('jdoe', 'mail.com', null);

INSERT INTO TABLE pageviews PARTITION (datestamp)

VALUES ('tjohnson', 'sports.com', 'finance.com', '2014-09-23'), ('tlee', 'finance.com', null, '2014-09-21');

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

You can set these configuration in hive-site.xml (after setting restart Hive) for ever or via terminal.

Dont Forget to restart Hive once the above settings are applied, else you will get the same error again.

2. Below query creates HiveTest table with ACID support

(To do Update,delete or Insert we need to create a table that support ACID properties)

```
CREATE TABLE HIVETEST
  (EMPLOYEEID INT,FIRSTNAME STRING,DESIGNATION STRING,
   SALARY INT,DEPARTMENT STRING)
  CLUSTERED BY (DEPARTMENT) INTO 3 BUCKETS
  STORED AS ORC TBLPROPERTIES ('TRANSACTIONAL'='TRUE') ;
```

3. Load data into HiveTest from a staging table,which contains the original data.

```
FROM STAGINGTBL
  INSERT INTO TABLE HIVETEST
  SELECT EMPLOYEEID,FIRSTNAME,DESIGNATION,SALARY,DEPARTMENT;
```

1.UPDATE

```
UPDATE HIVETEST
   SET SALARY = 50000
   WHERE EMPLOYEEID = 19;
```

SYNOPSIS

1. The referenced column must be a column of the table being updated.

2. The value assigned must be an expression that Hive supports in the select clause. Thus arithmetic operators, UDFs, casts, literals, etc. are supported. Subqueries are not supported.

3. Only rows that match the WHERE clause will be updated.

4. Partitioning columns cannot be updated.

5. Bucketing columns cannot be updated.

6. In Hive 0.14, upon successful completion of this operation the changes will be auto-committed.

2. INSERT

```
INSERT INTO TABLE HIVETEST
   VALUES(21,'HIVE','HIVE',0,'B');
```

SYNOPSIS

1. Each row listed in the VALUES clause is inserted into table tablename.

2. Values must be provided for every column in the table. The standard SQL syntax that allows the user to insert values into only some columns is not yet supported. To mimic the standard SQL, nulls can be provided for columns the user does not wish to assign a value to.

3. Dynamic partitioning is supported in the same way as for INSERT...SELECT.

4. If the table being inserted into supports ACID and a transaction manager that supports ACID is in use, this operation will be auto-committed upon successful completion.

3. DELETE

```
DELETE FROM HIVETEST
   WHERE EMPLOYEEID=19;
```

==========

FILE FORMAT

==========

```
-- Latest syntax using stored as avro
-- Run the sqoop import
sqoop import-all-tables \
 -m 12 \
 --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
 --username=retail_dba \
 --password=cloudera \
 --as-avrodatafile \
 --warehouse-dir=/user/hive/warehouse/retail_stage.db
-- It will create directories under warehouse-dir and copy data to that location
-- The command will also generate avsc files for each of the table with sqoop_import_<table_name>.avsc
-- Create directory in hdfs /user/cloudera/retail_stage
-- Copy all avsc files using hadoop fs -put /<path>/*.avsc /user/cloudera/retail_stage
-- Now you can create table in retail_stage db for all the data you have copied
use retail_stage;
CREATE EXTERNAL TABLE categories
STORED AS AVRO
LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/categories'
TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_categories.avsc');
```

```
CREATE EXTERNAL TABLE customers

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/customers'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_customers.avsc');

CREATE EXTERNAL TABLE departments

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/departments'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_departments.avsc');

CREATE EXTERNAL TABLE orders

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/orders'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_orders.avsc');

CREATE EXTERNAL TABLE order_items

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/order_items'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_order_items.avsc');

CREATE EXTERNAL TABLE products

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/products'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_products.avsc');

CREATE TABLE orders_part_avro (

order_id int,

order_date bigint,

order_customer_id int,

order_status string

)PARTITIONED BY (order_month string)

STORED AS AVRO
```

```
LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/orders_part_avro'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/orders_part_avro.avsc');

-- Adding partition manually

alter table orders_part_avro add partition (order_month='2014-01');

-- Inserting data to a partition

insert into table orders_part_avro partition (order_month='2014-01')

select * from orders where from_unixtime(cast(substr(order_date, 1, 10) as int)) like '2014-01%';

-- Drop table and recreate to test dynamic insert

-- Dynamic insert

set hive.exec.dynamic.partition.mode=nonstrict;

insert into table orders_part_avro partition (order_month)

select order_id, order_date, order_customer_id, order_status,

substr(from_unixtime(cast(substr(order_date, 1, 10) as int)), 1, 7) order_month from orders;

--validate

dfs -ls /user/hive/warehouse/retail_stage.db/orders_part_avro/*

dfs -ls /user/hive/warehouse/retail_stage.db/orders_part_avro/

==========

CREATE EXTERNAL TABLE categories

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'

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/categories'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_categories.avsc');

CREATE EXTERNAL TABLE customers

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'

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/customers'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_customers.avsc');
```

```
CREATE EXTERNAL TABLE departments

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'

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/departments'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_departments.avsc');
```

```
CREATE EXTERNAL TABLE orders

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'

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/orders'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_orders.avsc');
```

```
CREATE EXTERNAL TABLE order_items

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'

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/order_items'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_order_items.avsc');
```

```
CREATE EXTERNAL TABLE products

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'

LOCATION 'hdfs:///user/hive/warehouse/retail_stage.db/products'

TBLPROPERTIES
('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail_stage/sqoop_import_products.avsc');
```

==========

########Copy data from MySQL to local file system

#Enable file_priv to retail_dba

mysql -u root -p #if password enabled, else "mysql -u root"

update mysql.user set file_priv = 'Y' where user = 'retail_dba';

commit;

exit;

#On OS prompt, run

service mysqld restart

mysql -u retail_dba -p #prompts for password and launches mysql CLI

use retail_db;

#Make sure you understand table structure, delimiter, partition etc, run mysql export command

select * from categories into outfile '/tmp/categories01.psv' fields terminated by '|' lines terminated by '\n';

select * from customers into outfile '/tmp/customers.psv' fields terminated by '|' lines terminated by '\n';

select * from departments into outfile '/tmp/departments.psv' fields terminated by '|' lines terminated by '\n';

select * from products into outfile '/tmp/products.psv' fields terminated by '|' lines terminated by '\n';

#We cannot use orders and order_items directly as tables in hive database retail_ods are partitioned

########Load data from local file system to hive table

load data local inpath '/tmp/categories01.psv' overwrite into table categories;

load data local inpath '/tmp/customers.psv' overwrite into table customers;

load data local inpath '/tmp/departments.psv' overwrite into table departments;

load data local inpath '/tmp/products.psv' overwrite into table products;

#You can remove overwrite while appending data to underlying hive table

########Load data from HDFS to hive table

#Prepare HDFS stage directory

#On command prompt (if you login as root)

hadoop fs -mkdir /user/root/departments

hadoop fs -put /tmp/departments.psv /user/root/departments

hadoop fs -ls /user/root/departments

```
#Launch hive
hive
use retail_ods;
load data inpath '/user/root/departments/*' overwrite into table departments;
hadoop fs -ls /user/root/departments
#You will not find files

#Prepare orders on mysql database
#on mysql
select * from orders into outfile '/tmp/orders.psv' fields terminated by '|' lines terminated by '\n';
#Create orders_stage under hive database retail_stage
hive
use retail_stage;
 CREATE TABLE orders_stage (
order_id int,
order_date string,
order_customer_id int,
order_status string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE;
load data local inpath '/tmp/orders.psv' overwrite into table orders_stage;
insert overwrite table retail_ods.orders partition (order_month)
select order_id, order_date, order_customer_id, order_status,
substr(order_date, 1, 7) order_month from retail_stage.orders_stage;

#Now we have 2 tables retail_stage.order_items and retail_stage.orders
#We need to join these 2 and populate retail_ods.order_items table which have additional columns
#order_item_order_date and order_month
#Also table is partitioned by order_month
INSERT OVERWRITE TABLE ORDER_ITEMS PARTITION (ORDER_MONTH)
```

```
SELECT OI.ORDER_ITEM_ID, OI.ORDER_ITEM_ORDER_ID, O.ORDER_DATE,
OI.ORDER_ITEM_PRODUCT_ID, OI.ORDER_ITEM_QUANTITY, OI.ORDER_ITEM_SUBTOTAL,
OI.ORDER_ITEM_PRODUCT_PRICE, SUBSTR(O.ORDER_DATE, 1, 7)
ORDER_MONTH FROM RETAIL_STAGE.ORDER_ITEMS OI JOIN RETAIL_STAGE.ORDERS_STAGE O
ON OI.ORDER_ITEM_ORDER_ID = O.ORDER_ID;

SELECT D.DEPARTMENT_NAME, SUBSTR(O.ORDER_DATE, 1, 7) MONTH,
SUM(OI.ORDER_ITEM_SUBTOTAL) MONTHLY_REVENUE
FROM ORDERS O JOIN ORDER_ITEMS OI ON OI.ORDER_ITEM_ORDER_ID = O.ORDER_ID
JOIN PRODUCTS P ON OI.ORDER_ITEM_PRODUCT_ID = P.PRODUCT_ID
JOIN CATEGORIES C ON P.PRODUCT_CATEGORY_ID = C.CATEGORY_ID
JOIN DEPARTMENTS D ON C.CATEGORY_DEPARTMENT_ID = D.DEPARTMENT_ID
WHERE O.ORDER_DATE LIKE '2013%'
GROUP BY D.DEPARTMENT_NAME, SUBSTR(O.ORDER_DATE, 1, 7);

CREATE TABLE RETAIL_DENORMALIZED
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
AS
SELECT D.*, OI.*, O.ORDER_DATE
FROM ORDERS O JOIN ORDER_ITEMS OI ON OI.ORDER_ITEM_ORDER_ID = O.ORDER_ID
JOIN PRODUCTS P ON OI.ORDER_ITEM_PRODUCT_ID = P.PRODUCT_ID
JOIN CATEGORIES C ON P.PRODUCT_CATEGORY_ID = C.CATEGORY_ID
JOIN DEPARTMENTS D ON C.CATEGORY_DEPARTMENT_ID = D.DEPARTMENT_ID;

SELECT DEPARTMENT_NAME, SUBSTR(ORDER_DATE, 1, 7) ORDER_MONTH, SUM(ORDER_ITEM_SUBTOTAL)
MONTHLY_REVENUE
FROM RETAIL_DENORMALIZED WHERE ORDER_DATE LIKE '2013%'
GROUP BY DEPARTMENT_NAME, SUBSTR(ORDER_DATE, 1, 7);
```