Web: Inceptez.com Mail: info@inceptez.com Call: 7871299810, 7871299817

INCEPTEZ TECHNOLOGIES HIVE WORKOUTS

HIVE Installation with local metastore configuration
----

1) copy the hive tar file to your home path (/home/hduser/install) and extract using below command
cd /home/hduser/install/
tar xvzf apache-hive-0.14.0-bin.tar.gz

sudo mv apache-hive-0.14.0-bin /usr/local/hive

2) Make an entry in the bash profile like below,
vi ~/.bashrc
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin

Then source the bash profile like below,

source ~/.bashrc

3) After you complete the above steps execute below commands to create directories and give permissions

hadoop fs -mkdir -p /user/hive/warehouse/
hadoop fs -chmod g+w /user/hive/warehouse
hadoop fs -chmod g+w /tmp

4) Cd to bin folder inside hive path

cd /usr/local/hive/bin

put an entry like below,

vi hive-config.sh

export HADOOP_HOME=/usr/local/hadoop

==========

A. Create Database
----
create database retail;

B. Select Database
----
use retail;

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

C. Create table for storing transactional records
----

creating managed tables:

```
create table txnrecords(txnno INT, txndate STRING, custno INT, amount DOUBLE,
category STRING, product STRING, city STRING, state STRING, spendby STRING)
row format delimited
fields terminated by ','
lines terminated by '\n'
stored as textfile;
```

!mkdir -p /home/hduser/hive/data;

Copy all content from data path in the pen drive to the above path.
/home/hduser/hive/data

D. Load the data into the table [From Linux client]

LOAD DATA LOCAL INPATH '/home/hduser/hive/data/txns' OVERWRITE INTO TABLE txnrecords;

Load the data into the table [From HDFS]

```
cd /home/hduser/hive/data/
hadoop fs -copyFromLocal txns txns1
```

LOAD DATA INPATH '/user/hduser/txns1' OVERWRITE INTO TABLE txnrecords;

Select the loaded data

```
Map/reduce execution check?
select count(1)  from txnrecords ;
select * from txnrecords limit 10;
select * from txnrecords where category='Puzzles';
```

Creating External tables:

```
create external table externaltxnrecords(txnno INT, txndate STRING, custno INT, amount DOUBLE,
category STRING, product STRING, city STRING, state STRING, spendby STRING)
row format delimited
fields terminated by ','
stored as textfile location '/user/hduser/hiveexternaldata';
```

E. Describing metadata or schema of the table

```
describe formatted txnrecords;
```

describe formatted externaltxnrecords;

Index
------
create index idx_custno on table txnrecords(custno) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;

Array Example
----

create table array(id int,name string,sal bigint,desig array<string>,city string)
row format delimited
fields terminated by ','
collection items terminated by '$';

Create file with below contents:

vi /home/hduser/hive/data/arraydata

data

1,Arvindh,40000,SWE$Analyst$SAnalyst$PL,hyd
2,Bala,30000,SWE$Analyst$SAnalyst$PM,hyd
3,Chandra,50000,SWE$Analyst$PL$PM,Che
4,Gokul,30000,SWE$Analyst$SAnalyst$,hyd

load data local inpath '/home/hduser/hive/data/arraydata' overwrite into table array;

select name,desig[2],sal from array;

Struct Example
----

create table Struct(id int,name string,sal bigint,addr struct<city:string,state:string,pin:bigint>)
row format delimited
fields terminated by ','
collection items terminated by '$';

vi /home/hduser/hive/data/struct

data

1,Arvindh,40000,HyderabadAP800042
2,Bala,30000,Chennai$TamilNadu$600042

load data local inpath '/home/hduser/hive/data/struct' overwrite into table Struct;

select addr.city from struct;

Map Example
----

create table map1 (id int,name string,sal bigint,Mark map<string,int>,city string)

row format delimited
fields terminated by ','
collection items terminated by '$'
map keys terminated by '#';

data

1,Arvindh,40000,mat#100$Eng#99,hyd
2,Bala,30000,Sci#100$Eng#99,chn

load data local inpath '/home/hduser/hive/data/map1' overwrite into table map1;

Select Mark["mat"] from map1;

I. Create partitioned table

External table with partition
----

create external table txnrecsByCat(txnno INT, txndate STRING, custno INT, amount DOUBLE,
product STRING, city STRING, state STRING, spendby STRING)
partitioned by (category STRING)
row format delimited
fields terminated by ','
stored as textfile
location '/user/hduser/hiveexternaldata';

Managed table with partition
----

create table managedtxnrecsByCat(txnno INT, txndate STRING, custno INT, amount DOUBLE,
product STRING, city STRING, state STRING, spendby STRING)
partitioned by (category STRING)
row format delimited
fields terminated by ','
stored as textfile;

hadoop fs -mkdir -p /user/hduser/hiveexternaldata

show partitions TXNRECSBYCAT;

K. Load data into partition table

Static Partition:

Managed table

Insert into table managedtxnrecsbycat partition (category='Games') select txnno,txndate,custno,amount,
product,city,state,spendby from txnrecords where category='Games';

External

Insert into table txnrecsbycat partition (category='Test') select txnno,txndate,custno,amount, product,city,state,spendby from txnrecords where category='Games';

Dynamic Partition:

J. Configure Hive to allow partitions

Run the below insert and see whether it is working

Insert into table txnrecsbycat partition (category) select txnno,txndate,custno,amount, product,city,state,spendby,category from txnrecords;

However, a query across all partitions could trigger an enormous MapReduce job if the table data and number of partitions are large. A highly suggested safety measure is putting Hive into strict mode, which prohibits queries of partitioned tables without a WHERE clause that filters on partitions. You can set the mode to nonstrict, as in the following session:

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

Insert into table txnrecsbycat partition (category) select txnno,txndate,custno,amount, product,city,state,spendby,category from txnrecords;

Bucketing

Now that you've seen it let's talk about how it works. Unlike partitioning where each value for the slicer or partition key gets its own space, in clustering a hash is taken for the field value and then distributed across buckets. In the example above, we created 10 buckets and are clustering on username. Each bucket then would contain multiple users, while restricting each username to a single bucket.

Create bucketed table over the partitioned table

Set the maximum number of reducers to the same number of buckets specified in the table metadata (i.e. 10)

```
set map.reduce.tasks = 10;
```

Use the following command set hive.enforce.bucketing = true; allows the correct number of reducers and the cluster by column to be automatically selected based on the table.

```
set hive.enforce.bucketing = true ;

create table buckettxnrecsByCat(txnno INT, txndate STRING, custno INT, amount DOUBLE,
product STRING, city STRING, state STRING, spendby STRING)
partitioned by (category STRING)
clustered by (state) INTO 10 buckets
row format delimited
fields terminated by ','
stored as textfile location '/user/hduser/hiveexternaldata/bucketout';
```

Insert into table buckettxnrecsByCat partition (category) select txnno,txndate,custno,amount, product,city,state,spendby,category from txnrecords;

select count(1),category from buckettxnrecsByCat group by category;

SELECT txnno,product,state FROM buckettxnrecsByCat TABLESAMPLE(BUCKET 3 OUT OF 10);

Export hive data to a file:
---

insert overwrite local directory '/home/hduser/exptxnrecords' row format delimited fields terminated by ','
select txnno,txndate,custno,amount,
product,city,state,spendby from txnrecords where category='Games';

SerDe
=======

Create a xml file: xmlserde1

```
<records>
  <record>
        <empid>82338</empid>
        <empname>Ashfaq</empname>
    <income>200000</income>
    <age>29</age>
  </record>
  <record>
        <empid>83993</empid>
        <empname>Vivek</empname>
    <income>300000</income>
    <age>33</age>
  </record>
  <record>
        <empid>83994</empid>
        <empname>Vishwa</empname>
    <income>200000</income>
    <age>33</age>
  </record>
  <record>
        <empid>83994</empid>
        <empname>Arun</empname>
    <income>200000</income>
    <age>30</age>
  </record>
 <record1>
    <empid>83994</empid>
    <empname>Arun</empname>
    <income>200000</income>
    <age>30</age>
  </record1>
```

```
</records>
<recs>
  <record2>
     <empid>83996</empid>
     <empname>Arun</empname>
     <income>200000</income>
     <age>30</age>
   </record2>
</recs>
```

Copy the serde to hive/lib

/usr/local/hive/lib/hivexmlserde-1.0.5.3.jar
and other 2 jars (mysql-connector-java.jar and hive-exec-0.14.0.jar)

create below table

```
create external table xml_Emp1(empid string, empname string,income BIGINT,age int)
row format serde 'com.ibm.spss.hive.serde2.xml.XmlSerDe'
WITH SERDEPROPERTIES (
"column.xpath.empid"="/record/empid/text()",
"column.xpath.empname"="/record/empname/text()",
"column.xpath.income"="/record/income/text()",
"column.xpath.age"="/record/age/text()"
)
STORED AS
INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
location '/user/hduser/xmlserde'
TBLPROPERTIES ("xmlinput.start"="<record>","xmlinput.end"="</record>");
```

Copy file to hdfs

```
hadoop fs -put /home/hduser/hive/data/xmlserde1 /user/hduser/xmlserde/

select * from xml_Emp1;
```

HIVE UDF:

UDF using java jar

==========

```
cp HiveUdf.jar /home/hduser/hive/

ADD JAR /home/hduser/hive/HiveUdf.jar;

create function hello as 'Inceptez.Training.Hello.helloword';

select hello(empname) from xml_Emp1;
```

==========
Remote Metastore configuration
==========
I) MYSQL DB changes:
==========
su root
Password : hduser

su mysql
service mysqld start
exit

mysql -u root -p
Enter password: root

create database metastore;
USE metastore;
SOURCE /usr/local/hive/scripts/metastore/upgrade/mysql/hive-schema-0.14.0.mysql.sql
show tables;

CREATE USER 'hiveuser'@'%' IDENTIFIED BY 'hivepassword';
GRANT all on *.* to 'hiveuser'@localhost identified by 'hivepassword';
flush privileges;

II) HIVE-SITE.XML configuration to connect to remote metastore
==========

Copy the hive-site_rms_transction.xml config file provided in the pen drive to /usr/local/hive/conf

cp /home/hduser/hive/hive-site_rms_transction.xml /usr/local/hive/conf/hive-site.xml

OR

Execute the below steps

cd /usr/local/hive/conf
mv hive-default.xml.template hive-site.xml

Make below changes in
vi /usr/local/hive/conf/hive-site.xml

```
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://localhost:9083</value>
  <description>IP address (or fully-qualified domain name) and port of the metastore host</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://localhost/metastore</value>
```

```
 <description>the URL of the MySQL database</description>
</property>

<property>
 <name>javax.jdo.option.ConnectionDriverName</name>
 <value>com.mysql.jdbc.Driver</value>
</property>

<property>
 <name>javax.jdo.option.ConnectionUserName</name>
 <value>hiveuser</value>
</property>

<property>
 <name>javax.jdo.option.ConnectionPassword</name>
 <value>hivepassword</value>
</property>

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

<property>
 <name>hive.stats.dbclass</name>
 <value>jdbc:mysql</value>
 <description>The default database that stores temporary hive statistics.</description>
</property>

<property>
 <name>hive.stats.jdbcdriver</name>
 <value>com.mysql.jdbc.Driver</value>
 <description>The JDBC driver for the database that stores temporary hive statistics.</description>
</property>

<property>
 <name>hive.stats.dbconnectionstring</name>
 <value>jdbc:mysql://localhost/hive?createDatabaseIfNotExist=true</value>
 <description>The default connection string for the database that stores temporary hive
statistics.</description>
</property>
```

Note : Before you start hive metastore service make sure you have copied all the 3 jar files to lib path

----

```
cp /home/hduser/hive/Hive_jars/hive-exec-0.14.0.jar hivexmlserde-1.0.5.3.jar mysql-connector-java.jar
/usr/local/hive/lib/
```

III) Running the services before starting hive session (Run this in a different terminal)

==========

```
hive --service metastore
```

IV) Now enter into hive shell (in a different terminal)

hive

```
==============================================
Performing DML statements in Hive.14
==============================================

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

1.Create HiveTest table with DML support**

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');

2. INSERT**

insert into table HiveTest
    values(1,'Rohit','MD',88000,'Development');

3. SELECT

SELECT * FROM hivetest;

4.UPDATE

 update HiveTest set salary = 111111 where employeeid = 1;

5. DELETE

 delete from HiveTest where employeeid=1;


========================
Hive Sqoop Integeration
========================
Create a hive table using sqoop as like mysql table
----------------------------------------------------

sqoop create-hive-table --connect jdbc:mysql://localhost/test --username hiveuser -P --table customer --hive-table customer;

Import data from sqoop directly to hive customer table
```

```
sqoop import --connect jdbc:mysql://localhost/test --username hiveuser -P --table customer --hive-import --hive-table default.customer -m 1
```

==========

==========

Few queries for self practice:

==========

find sales based on age group

==========

```
create table customer(custno string, firstname string, lastname string, age int,profession string)
row format delimited
fields terminated by ',';

load data local inpath '/home/hduser/hive/data/custs' into table customer;
```

Using a simple join on 2 tables:

==========

```
create table cust_trxn (custno int,firstname string,age int,profession string,amount double,product string)
row format delimited
fields terminated by ',';

insert into table cust_trxn
select a.custno,a.firstname,a.age,a.profession,b.amount,b.product
from customer a JOIN txnrecords b ON a.custno = b.custno;

select * from cust_trxn limit 10;
```

Using CASE statement

==========

```
create table cust_trxn_case (custno int,firstname string,age int,profession string,amount double,product string, level string)
row format delimited
fields terminated by ',';

insert overwrite table cust_trxn_case
select * , case
 when age<30 then 'low'
 when age>=30 and age < 50 then 'middle'
 when age>=50 then 'old'
 else 'others'
end
from cust_trxn;

select * from cust_trxn_case limit 100;
```

Using Aggregate function
==========

```
create table cust_trxn_aggr (level string, amount double)
row format delimited
fields terminated by ',';

insert overwrite table cust_trxn_aggr
select level,sum(amount) from cust_trxn_case group by level;
```

UDF using Python code:
==========

```
CREATE TABLE u_data ( userid INT, movieid INT, rating INT, unixtime STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;

1       101     8       1369721454
2       102     8       1369821454
3       103     8       1369921454
4       105     8       1370021454
5       106     9       1370021454
```

And load it into the table that was just created:

```
LOAD DATA LOCAL INPATH '/home/hduser/udfdata' OVERWRITE INTO TABLE u_data;


SELECT COUNT(*) FROM u_data;
```

Create weekday_mapper.py:

```
import sys
import datetime
for line in sys.stdin:
        line = line.strip()
        userid, movieid, rating, unixtime = line.split('\t')
        weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
        print '\t'.join([userid, movieid, rating, str(weekday)])



CREATE TABLE u_data_new (userid INT, movieid INT, rating INT, weekday INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';

add FILE /usr/local/hive/examples/weekday_mapper.py;
```

Note : columns will be transformed to string and delimited by TAB before feeding to the user script, and the standard output of the user script will be treated as TAB-separated string columns.

The following command uses the TRANSFORM clause to embed the mapper scripts.

```
INSERT OVERWRITE TABLE u_data_new
```

```
SELECT
TRANSFORM (userid, movieid, rating, unixtime)
USING 'python weekday_mapper.py'
AS (userid, movieid, rating, weekday)
FROM u_data;

SELECT weekday, COUNT(*)
FROM u_data_new
GROUP BY weekday;
```