**when etc/hosts file is modified run the command to update network**

:service network restart

Mapreduce input format conf setting

key value input format:

conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", ",");

by default key and value are set to "tab" \t

NLine input format:

conf.set( "mapreduce.input.lineinputformat.linespermap ","n");

n= no of lines you want to assign to nline input format. and n should be of type string.

**hive table creation:**

**# desc formatted tableName**

**(to get detailed information of table)**

**# describe table name**

**(schema description of table)**

I) Managed table

create table TableName (ColumnName Datatype, column2 Datatype)

row format delimited fields terminated by '\t';

II)External Table

Create External table TableName (ColumnName Datatype, column2 Datatype)

row format delimited fields terminated by '\t'

location '/hdfs path' ;

Loading data from local file system to hive table:

# Load data local inpath '/path of file' into table Tablename;

if loading data from hdfs location:

# Load data inpath '/path of file' into table Tablename;

**Creating partition table**:

Create table Tablename (ColumnName datatypes) partitioned by (ColumnName Datatype) row format delimited fields terminated by '\t';

loading data into partition table statically:

#Load data local inpath '/path of file' into table TableName partition(columnName='value');

// it creates sub directory with name as "columnname=value" and stores the actual file into that directory

Loading data into partition table dynamically:

// this can be done only when the data is already present in the table and we are loading it into the partitioned table

# insert overwrite table TableName partition(ColumnName)

Query;

ex:

insert overwrite table dynpart partition(empid) select \* from manage1;

**writing hive udf in java**

download hive-exec jar from

http://mvnrepository.com/artifact/org.apache.hive/hive-exec/0.12.0

* add the jar to eclipse
* create a class that extends UDF
* overwrite the evaluate() method

**package** com.userfun;

**import** org.apache.hadoop.hive.ql.exec.UDF;

**import** org.apache.hadoop.io.Text;

**public** **class** userfunction **extends** UDF{

**public** Text evaluate(Text input)

{

StringBuffer sb=**new** StringBuffer();

String str=input.toString();

String[] st=str.split("-");

sb.append(st[0]+"/");

sb.append(st[1]+"/");

sb.append(st[2]);

**return** **new** Text(sb.toString());

}

}

* in the hive terminal do the following
* add jar (path of the jar file)
* # add jar /home/cloudera/Desktop/hiveudf2.jar ;
* create temporary function (function name) as 'fully qualified name';
* # create temporary function dateudf1 as 'com.userfun.userfunction'
* fully qualified name mean :
* packagename.classname

Hadoop Pig Macros

grunt > define **max\_shipid**(x,cust\_id,shipid) returns y

{

A = group $x by $cust\_id;

$y = foreach A generate group,MAX($x.$shipid);

};

grunt> a = load 'path to file' using PigStorage()

> as (feild name : data\_type,....);

grunt > max\_ship = max\_shipid(a,customerid,shipid);

the above macro gives you maximum value of shipid for eacg customerid;

the above macro takes 3 argument as input

max\_shipid(a,customerid,shipid) here

a goes to x in macro

customerid goes to cust\_id in macro

shipid goes to shipid in macro

**writing UDF in PIG**

**download and add pig jar**

**1) filter udf**

**it is used for queries that consist FILTER in it**

**package com.stringudf;**

**import java.io.IOException;**

**import javax.print.DocFlavor.CHAR\_ARRAY;**

**import org.apache.pig.FilterFunc;**

**import org.apache.pig.data.Tuple;**

**//import java.util.ArrayList;**

**//import java.util.List;**

**//import org.apache.pig.FilterFunc;**

**//import org.apache.pig.backend.executionengine.ExecException;**

**//import org.apache.pig.data.DataType;**

**//import org.apache.pig.data.Tuple;**

**//import org.apache.pig.impl.logicalLayer.FrontendException;**

**public class stringudf extends FilterFunc {**

**@Override**

**public Boolean exec(Tuple arg0) throws IOException {**

**// TODO Auto-generated method stub**

**Object object = arg0.get(0);**

**String id = (String)object;**

**String s1 = "1996-07-04";**

**return id.equals(s1) || id.equals("2004-12-12");**

**}**

**}**

**grunt>register jar name;**

**execute query ;**

**query :**

filt = filter a by packagename.classname(feild name );

in case you want to filter the data present in table a by its feild orderdate then :

filt = filter a by packagename.classname(orderdate);

or

filt = filter a by aliase(orderdate);

if you do not want to write package name.class name then

grunt> define aliase packagename.classname();

**eval udf:**

**//used with query that consist of "for each" condition that generates feilds.**

package com.evaludf;

import java.io.IOException;

import org.apache.pig.EvalFunc;

import org.apache.pig.data.Tuple;

public class evaludf extends EvalFunc<String>{

@Override

public String exec(Tuple arg0) throws IOException {

// TODO Auto-generated method stub

Object object = arg0.get(0);

String str = (String)object;

String[] split = str.split("-");

return split[0];

}

}

query:

gen = foreach a generate packagename.classname(orderdate);