**--Create Database**

CREATE DATABASE IF NOT EXISTS hive\_db\_demo

COMMENT 'weclouddata hive database'

WITH DBPROPERTIES ('creator'='weclouddata','date'='2017-03-03');

**--Describe and Use Database**

SHOW DATABASES;

DESCRIBE DATABASE hive\_db\_demo;

USE hive\_db\_demo;

**--Drop the Database**

DROP DATABASE IF EXISTS hive\_db\_demo;

**--Create Customer Table**

CREATE TABLE IF NOT EXISTS customer(

id string,

credits ARRAY<string>,

profile STRUCT<name:string,gender:string,dob:date,height:int,citizenship:string,student:boolean>,

contact MAP<string,string>,

address STRUCT<no:string,street:string,city:string,province:string,country:string,postal:string>,

preferpayment string

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '$'

COLLECTION ITEMS TERMINATED BY '^'

MAP KEYS TERMINATED BY '#';

**--Prepare Data File and Save into /root/TrainingOnHDP/dataset/customer.txt**

jiangb$visa^master^debit^cash$bin jiang^male^2010-01-01^170^canada^true$cell#4161111010^home#9051011111^office#4168888888$100^highway 7^markham^on^canada^l155j1$visa

michael$master^debit^cash$michael liu^male^1990-05-05^180^canada^false$cell#6471111888^home#4165011111$888^sheppard ave^scarborough^on^canada^m1l5k1$cash

tina$visa^master^debit$tina fan^female^2000-01-09^160^china^false$cell#4161871010$100^16th ave^richmond hill^on^canada^k155j1$debt

ryan$cash$ryan witcom^male^1980-09-05^185^france^false$home#9056011111$3000^victoria park ave^north york^on^canada^g1l5l1$master

**--Load Data into Customer Table**

LOAD DATA LOCAL INPATH '/root/TrainingOnHDP/dataset/customer.txt' OVERWRITE INTO TABLE customer;

**--Query the Customer table**

SELECT \* FROM customer;

**--Query the ARRAY in the Customer table**

SELECT credits FROM customer;

SELECT credits[0] AS credit0, credits[1] AS credit1, credits[2] AS credit2, credits[3] AS credit3 FROM customer;

**--Show Column Name**

set hive.cli.print.header=true;

**--Query the STRUCT in the Customer table**

SELECT profile FROM customer;

SELECT profile.name, profile.gender, profile.dob, profile.height, profile.citizenship, profile.student FROM customer;

SELECT address FROM customer;

SELECT id, address.no, address.street, address.city, address.province, address.country, address.postal FROM customer;

**--Query the MAP in the Customer table**

SELECT contact FROM customer;

SELECT id, contact['cell'] AS cell\_phone, contact['home'] AS home\_phone, contact['office'] AS office\_phone FROM customer;

**--Create Orders Table**

CREATE TABLE IF NOT EXISTS orders(

id string,

userid string,

vender string,

amount double,

quantity int

)

PARTITIONED BY (city string, purchasedate string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

**--Insert Data into Orders table**

INSERT INTO orders partition(city, purchasedate) VALUES ('00001', 'jiangb', 'walmart', 100.00, 5, 'toronto', '20160501');

INSERT INTO orders partition(city, purchasedate) VALUES ('00002', 'jiangb', 'loblaws', 300.00, 15, 'toronto', '20170303');

INSERT INTO orders partition(city, purchasedate) VALUES ('00003', 'michael', 'apple', 55.10, 3, 'paris', '20150607');

INSERT INTO orders partition(city, purchasedate) VALUES ('00004', 'tina', 'walmart', 6000.00, 55, 'toronto', '20160501');

INSERT INTO orders partition(city, purchasedate) VALUES ('00005', 'ryan', 'loblaws', 700.00, 7, 'toronto', '20170203');

INSERT INTO orders partition(city, purchasedate) VALUES ('00006', 'tina', 'apple', 155.10, 1, 'paris', '20170707');

INSERT INTO orders partition(city, purchasedate) VALUES ('00007', 'jiangb', 'walmart', 100.00, 2, 'toronto', '20160501');

INSERT INTO orders partition(city, purchasedate) VALUES ('00008', 'jiangb', 'loblaws', 990.00, 12, 'toronto', '20170303');

INSERT INTO orders partition(city, purchasedate) VALUES ('00009', 'michael', 'apple', 555.10, 9, 'paris', '20150607');

INSERT INTO orders partition(city, purchasedate) VALUES ('000010', 'tina', 'walmart', 80.00, 6, 'toronto', '20160501');

INSERT INTO orders partition(city, purchasedate) VALUES ('000011', 'ryan', 'loblaws', 7.00, 1, 'toronto', '20170203');

INSERT INTO orders partition(city, purchasedate) VALUES ('000012', 'tina', 'apple', 378.10, 8, 'paris', '20170707');

**--Query the Orders table**

SELECT \* FROM orders;

**--Show Orders table partitions**

SHOW PARTITIONS orders;

**--Add partitions to Orders**

ALTER TABLE orders ADD

PARTITION (city='montreal', purchasedate='20000101')

PARTITION (city='montreal', purchasedate='20010909');

**--Drop partitions from Orders**

ALTER TABLE orders DROP PARTITION (city='montreal', purchasedate='20000101');

ALTER TABLE orders DROP PARTITION (city='montreal', purchasedate='20010909');

**--Prepare Data File and save into /root/orders.txt**

000013,tina,walmart,765,6

000014,tina,walmart,765,6

000015,tina,walmart,765,6

**--Upload into HDFS /tmp/orders.txt**

hadoop fs -copyFromLocal /root/TrainingOnHDP/dataset/orders.txt /tmp/orders.txt

hadoop fs -chmod 777 /tmp/orders.txt

**--Load data to the partition**

LOAD DATA INPATH '/tmp/orders.txt' INTO TABLE orders PARTITION (city='montreal', purchasedate='20000101');

**--Create a Orders History table with bucketing**

CREATE TABLE IF NOT EXISTS orders\_history(

id string,

userid string,

vender string,

amount double,

quantity int

)

PARTITIONED BY (city string, purchasedate string)

CLUSTERED BY (userid) INTO 2 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

SET hive.enforce.bucketing = true;

INSERT OVERWRITE TABLE orders\_history partition(city, purchasedate) SELECT \* FROM orders;

**--Dynamic partition is not enabled by default**

SET hive.exec.dynamic.partition=true;

SET hive.exec.dynamic.partition.mode=nostrict;

**--Prepare Data File and save into /root/customer\_action.txt**

jiangb,jsessionid000000000001,purchase,2017-03-03 00:17:13,chrome,google

jiangb,jsessionid000000000001,product,2017-03-03 00:17:33,chrome,google

michael,jsessionid000000000003,product,2017-03-03 00:18:33,ie,yahoo

tina,jsessionid000000000005,home,2017-03-03 00:19:33,firefox,facebook

ryan,jsessionid000000000006,info,2017-03-03 00:19:50,safari,linkedin

jiangb,jsessionid000000000011,shoppingcart,2017-03-03 00:19:33,chrome,google

**--Upload into HDFS /tmp/customeraction/edate=20170303**

hadoop fs -mkdir /tmp/customeraction

hadoop fs -mkdir /tmp/customeraction/edate=20170303

hadoop fs -copyFromLocal /root/customer\_action.txt /tmp/customeraction/edate=20170303/customer\_action.txt

hadoop fs -chmod -R 777 /tmp/customeraction

**--Create Customer Action external table and load the data**

CREATE EXTERNAL TABLE IF NOT EXISTS customer\_action(

userid string,

sessionid string,

page string,

visitedtime timestamp,

browser string,

referer string

)

partitioned by(edate string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/tmp/customeraction';

**--Refresh Custom Action table with new partition**

MSCK REPAIR TABLE customer\_action;

**--Prepare Data File and save into /root/customer\_action.txt**

jiangb,jsessionid100000000001,purchase,2017-03-02 00:17:13,chrome,google

jiangb,jsessionid100000000001,product,2017-03-02 00:17:33,chrome,google

michael,jsessionid100000000003,product,2017-03-02 00:18:33,ie,yahoo

tina,jsessionid100000000005,home,2017-03-02 00:19:33,firefox,facebook

ryan,jsessionid100000000006,info,2017-03-02 00:19:50,safari,linkedin

jiangb,jsessionid100000000011,shoppingcart,2017-03-02 00:19:33,chrome,google

**--Upload into HDFS /tmp/customeraction/edate=20170302**

hadoop fs -mkdir /tmp/customeraction/edate=20170302

hadoop fs -copyFromLocal /root/TrainingOnHDP/dataset/customer\_action.txt /tmp/customeraction/edate=20170302/customer\_action.txt

**--Refresh Custom Action table with new partition**

MSCK REPAIR TABLE customer\_action;

**--JOIN between two tables**

SELECT customer.id as userid, customer.profile.name as customer\_name, orders.id as orderid FROM customer JOIN orders ON customer.id = orders.userid;

SELECT customer.id as userid, customer.profile.name as customer\_name, orders.id as orderid FROM customer JOIN orders ON customer.id = orders.userid sort by orderid;

SELECT customer.id as userid, customer.profile.name as customer\_name, orders.id as orderid FROM customer JOIN orders ON customer.id = orders.userid order by orderid;

SELECT customer.id as userid, customer.profile.name as customer\_name, count(orders.id) as order\_number, sum(orders.amount) as total\_amount FROM customer JOIN orders ON customer.id = orders.userid group by customer.id, customer.profile.name;

SELECT customer.id as userid, customer.profile.name as customer\_name, collect\_set(orders.id), count(orders.id) as order\_number, sum(orders.amount) as total\_amount FROM customer JOIN orders ON customer.id = orders.userid group by customer.id, customer.profile.name;

**--MAP JOIN enabled by query hint**

SELECT /\*+ MAPJOIN(customer)\*/ customer.id, customer\_action.\* FROM customer JOIN customer\_action WHERE customer.id = customer\_action.userid;

**--Create Table With LIKE**

CREATE TABLE customer\_like LIKE customer;

DESCRIBE customer\_like;

INSERT INTO customer\_like

SELECT 'peter' as id, array('visa') as credits, named\_struct('name','peter sampas','gender','male','dob',CAST('1995-01-09' as date),'height',190,'citizenship','usa','student',false) as profile,

map('cell','4161506788') as contact,

named\_struct('no','999','street','eglinton ave','city','toronto','province','on','country','canada','postal','m1p5l1') as address, 'visa' as preferpayment

FROM customer limit 1;

**--Create Table With CTAS**

CREATE TABLE customer\_ctas AS SELECT \* FROM customer;

**--Create Table with Common Table Expression (CTE)**

CREATE TABLE customer\_cte AS

WITH male AS (SELECT profile.name as name FROM customer WHERE profile.gender= 'male' and id = 'jiangb'),

female AS (SELECT profile.name as name FROM customer WHERE profile.gender= 'female')

SELECT \* FROM male UNION ALL select \* FROM female;

**--Alter Table File Format to Parquet**

CREATE TABLE IF NOT EXISTS orders\_parquet(

id string,

userid string,

vender string,

amount double,

quantity int

)

PARTITIONED BY (city string, purchasedate string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS PARQUET;

INSERT INTO orders\_parquet partition(city,purchasedate) select \* from orders;

**--Create Hive view**

CREATE VIEW orders\_view AS select id, userid from orders;

--Nest SELECT

SELECT \* FROM (

SELECT \* FROM customer

WHERE profile.gender = 'male'

) male;

**--Subquery**

SELECT \* FROM orders WHERE orders.userid IN (SELECT id FROM customer WHERE customer.profile.gender = 'female');

**--UNION**

SELECT customer.id as username FROM customer UNION ALL SELECT orders.userid as username FROM orders;

SELECT DISTINCT a.username FROM (SELECT customer.id as username FROM customer UNION ALL SELECT orders.userid as username FROM orders) a;

**--INTERCEPT**

SELECT customer.id as username FROM customer JOIN orders ON customer.id = orders.userid;

**--MINUS**

SELECT customer.id as username FROM customer LEFT JOIN orders ON customer.id = orders.userid WHERE orders.userid IS NULL;

SELECT customer.id as username FROM customer LEFT JOIN orders ON customer.id = orders.userid WHERE orders.userid IS NULL

UNION ALL

SELECT orders.userid as username FROM customer RIGHT JOIN orders ON customer.id = orders.userid WHERE customer.id IS NULL;

**--INSERTS By Scanning the table only once**

FROM customer

INSERT OVERWRITE TABLE customer\_ctas

SELECT \* WHERE profile.gender = 'male'

INSERT OVERWRITE TABLE customer\_like

SELECT \* WHERE profile.gender = 'female';

**--Insert to local files with default row separators**

INSERT OVERWRITE LOCAL DIRECTORY '/root/customer\_output' SELECT \* FROM customer;

FROM customer

INSERT OVERWRITE LOCAL DIRECTORY '/root/customer\_output1'

SELECT \*

INSERT OVERWRITE LOCAL DIRECTORY '/root/customer\_output2'

SELECT \*;

**--Aggregation without GROUP BY columns**

SELECT count(\*) AS orders FROM orders;

**--HAVING**

SELECT userid, count(\*) as total\_order FROM orders GROUP BY userid HAVING count(\*) > 1;

**--GROUPING SETS**

SELECT city, purchasedate, userid, count(id) total\_order FROM orders

GROUP BY city, purchasedate, userid GROUPING SETS (city, purchasedate, userid);

**--ROLLUP**

SELECT city, purchasedate, userid, count(id) total\_order FROM orders

GROUP BY city, purchasedate, userid WITH ROLLUP;

**--CUBE**

SELECT city, purchasedate, userid, count(id) total\_order FROM orders

GROUP BY city, purchasedate, userid WITH CUBE;

**--Analytic Functions**

SELECT city, userid, id as orderid, amount,

COUNT(\*) OVER (PARTITION BY city) AS order\_bycity,

SUM(amount) OVER(PARTITION BY city ORDER BY city) AS amount\_bycity,

SUM(amount) OVER(ORDER BY city) AS total\_amount1,

SUM(amount) OVER(ORDER BY city, userid rows unbounded preceding) AS total\_amount2

FROM orders

ORDER BY city, userid;

SELECT city, userid, id as orderid, amount,

RANK() OVER (PARTITION BY city ORDER BY amount) AS rank,

DENSE\_RANK() OVER (PARTITION BY city ORDER BY amount) AS dense\_rank,

ROW\_NUMBER() OVER () AS row\_num,

ROUND((CUME\_DIST() OVER (PARTITION BY city ORDER BY amount)), 1) AS cume\_dist,

PERCENT\_RANK() OVER(PARTITION BY city ORDER BY amount) AS percent\_rank

FROM orders

ORDER BY city;

SELECT city, userid, id as orderid, amount,

LEAD(amount, 2) OVER(PARTITION BY city ORDER BY amount) AS lead,

LAG(amount, 2, 0) OVER(PARTITION BY city ORDER BY amount) AS lag,

FIRST\_VALUE(amount) OVER (PARTITION BY city ORDER BY amount) AS first\_value,

LAST\_VALUE(amount) OVER (PARTITION BY city ORDER BY amount) AS last\_value\_default,

LAST\_VALUE(amount) OVER (PARTITION BY city ORDER BY amount RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)

AS last\_value FROM orders ORDER BY city;

SELECT city, userid, id as orderid, amount,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) w1,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN 2 PRECEDING AND UNBOUNDED FOLLOWING) w2,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN 1 PRECEDING AND 2 FOLLOWING) w3,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) w4,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) w5,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN CURRENT ROW AND CURRENT ROW) w6,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) w7,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) w8,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) w9,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN UNBOUNDED PRECEDING AND 1 FOLLOWING) w10,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) w11,

MIN(amount) OVER (PARTITION BY city ORDER BY userid ROWS 2 PRECEDING) w12

FROM orders ORDER BY city, userid;

**--EXPLAIN statement**

EXPLAIN SELECT purchasedate, count(\*) FROM orders WHERE city='toronto' GROUP BY purchasedate;

**--ANALYZE statement**

ANALYZE TABLE customer COMPUTE STATISTICS;

ANALYZE TABLE orders PARTITION(city='toronto', purchasedate='20160501') COMPUTE STATISTICS;

**--Data file optimization**

SET hive.exec.compress.output=true;

SET io.seqfile.compression.type=BLOCK;

**--Compression**

SET hive.exec.compress.intermediate=true;

SET hive.intermediate.compression.codec=org.apache.hadoop.io.compress.SnappyCodec;

SET hive.exec.compress.output=true;

SET mapred.output.compression.codec=org.apache.hadoop.io.compress.SnappyCodec;

--Storage optimization

SET hive.exec.mode.local.auto=true;

SET hive.exec.mode.local.auto.inputbytes.max=50000000;

SET hive.exec.mode.local.auto.input.files.max=5;

**--JVM reuse**

SET mapred.job.reuse.jvm.num.tasks = 5;

**--Parallel running job**

SET hive.exec.parallel=true;

SET hive.exec.parallel.thread.number=16;

**--Map Join**

SET hive.auto.convert.join=true;

SET hive.mapjoin.smalltable.filesize=600000000;

SET hive.auto.convert.join.noconditionaltask = true;

SET hive.auto.convert.join.noconditionaltask.size = 10000000;

**--Skew Join**

SET hive.optimize.skewjoin=true;

SET hive.skewjoin.key=100000;

**--Use Cases 1: Calculate total number of sessions group by access date, landing and exit Page**

SELECT edate, landPage, exitPage, COUNT(DISTINCT sessionid) as total\_sessions

FROM ( SELECT sessionid, edate,

first\_value(page) over (partition by sessionid) as landPage,

last\_value(page) over (partition by sessionid) as exitPage

FROM ( SELECT page, edate, sessionid, visitedtime,

count(\*) over (PARTITION BY sessionid) as c,

rank() over (PARTITION BY sessionid order by visitedtime asc) as r

FROM customer\_action ) a

WHERE r = 1 or r = c ) b

GROUP BY edate, landPage, exitPage;

**--Use Case 2: Calculate total number of sessions which only access single page**

SELECT page, edate, sum(case when sc=1 then 1 else 0 end) as singlepage\_count, count(1) as total\_sessions

FROM ( SELECT sessionid, page, edate, visitedtime,

count(\*) over (PARTITION BY sessionid) as sc,

rank() over (PARTITION BY sessionid order by visitedtime asc) as srank

FROM customer\_action) t

WHERE srank = 1

GROUP by page, edate;

**--Use Case 3: Calculate total number of new users and repeat users by access date**

SELECT edate, sum(case when previous\_c=1 then 1 else 0 end) new\_users, sum(case when previous\_c>1 then 1 else 0 end) repeat\_users

FROM ( SELECT userid, edate, current\_c,

count(\*) over (PARTITION BY userid order by edate ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) previous\_c

FROM ( SELECT userid, edate,

count(\*) over (PARTITION BY userid, edate) as current\_c,

rank() over (PARTITION BY userid, edate order by visitedtime) as rank

FROM customer\_action ORDER BY userid, edate) p1

WHERE rank = 1

ORDER BY userid, edate) p2

GROUP BY edate ORDER BY edate;

**--Use Case 4: Calculate the path to purchase grouby by access date and session**

SELECT edate, sessionid, collect\_set(page) as path\_to\_purchase

FROM ( SELECT sessionid, edate, page,

last\_value(page) over(PARTITION BY sessionid) as last\_page

FROM customer\_action)a

WHERE last\_page = 'product'

GROUP BY edate, sessionid;

**--Use Case 5: Calculate most frequent next action for the users**

SELECT page, next\_page, c

FROM( SELECT sessionid, page,

lead(page,1) OVER (PARTITION BY sessionid order by visitedtime asc) as next\_page,

count(\*) OVER (PARTITION BY sessionid order by visitedtime asc) as c,

rank() OVER (PARTITION BY sessionid order by visitedtime asc) as page\_view

**FROM customer\_action) a**

**--Use Case 6: Predict the airline delay**

**1 Data Preparation**

**1.1 Upload the data file from sandbox into HDFS**

hadoop fs -mkdir /tmp/airflightsdelays/flight2007

hadoop fs -put /root/TrainingOnHDP/dataset/flights\_2007.csv /tmp/airflightsdelays/flight2007/flights\_2007.csv

hadoop fs -mkdir /tmp/airflightsdelays/flight2008

hadoop fs -put /root/TrainingOnHDP/dataset/flights\_2008.csv /tmp/airflightsdelays/flight2008/flights\_2008.csv

hadoop fs -mkdir /tmp/airflightsdelays/weather2007

hadoop fs -put /root/TrainingOnHDP/dataset/weather\_2007.csv /tmp/airflightsdelays/weather2007/weather\_2007.csv

hadoop fs -mkdir /tmp/airflightsdelays/weather2008

hadoop fs -put /root/TrainingOnHDP/dataset/weather\_2008.csv /tmp/airflightsdelays/weather2008/weather\_2008.csv

**1.2 Create and populate the tables**

CREATE EXTERNAL TABLE IF NOT EXISTS flight\_2007(

year int,

month int,

day int,

week int,

deptime string,

crsdeptime string,

arrtime string,

crsarrtime string,

uniquecarrier string,

flightnum string,

tailnum string,

actualelapsedtime int,

crselapsedtime int,

airtime int,

arrdelay int,

depdelay int,

origin string,

dest string,

distance int,

taxiin int,

taxiout int,

cancelled int,

cancelledcode string,

Diverted int,

CarrierDelay int,

WeatherDelay int,

NASDelay int,

SecurityDelay int,

LateAircraftDel int

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/tmp/airflightsdelays/flight2007'

tblproperties ("skip.header.line.count"="1");

CREATE TABLE flight\_2007\_enrich AS

select concat(year, if(month<10, concat(0,month), month), if(day<10, concat(0,day), day)) as fdate, year, month, day, week, substr(crsdeptime,1,2) as crshour, (case when depDelay > 15 then 1 else 0 end) as status, crsdeptime, depdelay, origin, dest, distance, cancelled from flight\_2007 where cancelled = 0 and origin = "ORD";

CREATE EXTERNAL TABLE IF NOT EXISTS flight\_2008(

year int,

month int,

day int,

week int,

deptime string,

crsdeptime string,

arrtime string,

crsarrtime string,

uniquecarrier string,

flightnum string,

tailnum string,

actualelapsedtime int,

crselapsedtime int,

airtime int,

arrdelay int,

depdelay int,

origin string,

dest string,

distance int,

taxiin int,

taxiout int,

cancelled int,

cancelledcode string,

Diverted int,

CarrierDelay int,

WeatherDelay int,

NASDelay int,

SecurityDelay int,

LateAircraftDel int

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/tmp/airflightsdelays/flight2008'

tblproperties ("skip.header.line.count"="1");

CREATE TABLE flight\_2008\_enrich AS

select concat(year, if(month<10, concat(0,month), month), if(day<10, concat(0,day), day)) as fdate, year, month, day, week, substr(crsdeptime,1,2) as crshour, (case when depDelay > 15 then 1 else 0 end) as status, crsdeptime, depdelay, origin, dest, distance, cancelled from flight\_2008 where cancelled = 0 and origin = "ORD";

CREATE EXTERNAL TABLE IF NOT EXISTS weather\_2007(

station string,

wdate string,

metrics string,

mvalue string,

c1 string,

c2 string,

c3 string,

c4 string

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/tmp/airflightsdelays/weather2007';

CREATE TABLE weather\_2007\_join AS

with tmin as (select wdate, mvalue as tmin from weather\_2007 where station = "USW00094846" and metrics = "TMIN"),

tmax as (select wdate, mvalue as tmax from weather\_2007 where station = "USW00094846" and metrics = "TMAX"),

prcp as (select wdate, mvalue as prcp from weather\_2007 where station = "USW00094846" and metrics = "PRCP"),

snow as (select wdate, mvalue as snow from weather\_2007 where station = "USW00094846" and metrics = "SNOW"),

awnd as (select wdate, mvalue as awnd from weather\_2007 where station = "USW00094846" and metrics = "AWND")

select tmax.wdate, tmax.tmax, tmin.tmin, prcp.prcp, snow.snow, awnd.awnd from tmin left join tmax on tmin.wdate = tmax.wdate left join prcp on tmin.wdate = prcp.wdate

left join snow on tmin.wdate = snow.wdate left join awnd on tmin.wdate = awnd.wdate;

CREATE EXTERNAL TABLE IF NOT EXISTS weather\_2008(

station string,

wdate string,

metrics string,

mvalue string,

c1 string,

c2 string,

c3 string,

c4 string

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/tmp/airflightsdelays/weather2008';

CREATE TABLE weather\_2008\_join AS

with tmin as (select wdate, mvalue as tmin from weather\_2008 where station = "USW00094846" and metrics = "TMIN"),

tmax as (select wdate, mvalue as tmax from weather\_2008 where station = "USW00094846" and metrics = "TMAX"),

prcp as (select wdate, mvalue as prcp from weather\_2008 where station = "USW00094846" and metrics = "PRCP"),

snow as (select wdate, mvalue as snow from weather\_2008 where station = "USW00094846" and metrics = "SNOW"),

awnd as (select wdate, mvalue as awnd from weather\_2008 where station = "USW00094846" and metrics = "AWND")

select tmax.wdate, tmax.tmax, tmin.tmin, prcp.prcp, snow.snow, awnd.awnd from tmin left join tmax on tmin.wdate = tmax.wdate left join prcp on tmin.wdate = prcp.wdate

left join snow on tmin.wdate = snow.wdate left join awnd on tmin.wdate = awnd.wdate;

**add jar /root/TrainingOnHDP/PredictionAirlineDelayOnHive/target/PredictionAirlineDelayOnHive-1.0-SNAPSHOT.jar;**

CREATE TEMPORARY FUNCTION calgap as 'ca.training.bigdata.hive.udf.hive.udf.CalHolidaysGap';

calgap UDF java code

package ca.training.bigdata.hive.udf.hive.udf

public final class CalHolidaysGap extends UDF {

String[] holidays = {"01/01/2007", "01/15/2007", "02/19/2007", "05/28/2007", "06/07/2007", "07/04/2007",

"09/03/2007", "10/08/2007", "11/11/2007", "11/22/2007", "12/25/2007",

"01/01/2008", "01/21/2008", "02/18/2008", "05/22/2008", "05/26/2008", "07/04/2008",

"09/01/2008", "10/13/2008", "11/11/2008", "11/27/2008", "12/25/2008"};

DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");

public LongWritable evaluate(final IntWritable year, IntWritable month, IntWritable day) {

Calendar start = Calendar.getInstance();

start.set(year.get(), month.get(), day.get());

long startTime = start.getTime().getTime();

long endTime = 0;

Date end = null;

long diffDays = 3000;

for (String holiday : holidays) {

try {

end = (Date) dateFormat.parse(holiday);

}catch(Exception e){}

endTime = end.getTime();

long diffTime = Math.abs(endTime - startTime);

if (diffDays > diffTime / (1000 \* 60 \* 60 \* 24))

diffDays = Math.round(diffTime / (1000 \* 60 \* 60 \* 24));

}

return new LongWritable(diffDays);

}

}

CREATE TABLE flight\_2007\_join AS

select a.status, a.month, a.day, a.week, cast(a.crshour as int) as crshour, a.distance, calgap(a.year, a.month, a.day) as gap, cast(b.tmax as int) as tmax, cast(b.tmin as int) as tmin, cast(b.prcp as int) as prcp, cast(b.snow as int) as snow, cast(b.awnd as int) as awnd from flight\_2007\_enrich a left join weather\_2007\_join b on a.fdate = b.wdate;

CREATE TABLE flight\_2008\_join AS

select a.status, a.month, a.day, a.week, cast(a.crshour as int) as crshour, a.distance, calgap(a.year, a.month, a.day) as gap, cast(b.tmax as int) as tmax, cast(b.tmin as int) as tmin, cast(b.prcp as int) as prcp, cast(b.snow as int) as snow, cast(b.awnd as int) as awnd from flight\_2008\_enrich a left join weather\_2008\_join b on a.fdate = b.wdate;

**add jar /root/TrainingOnHDP/lib/hivemall-core-0.4.2-rc.2-with-dependencies.jar;**

**source /root//TrainingOnHDP/libdefine-all.hive;**

**2 Data Normization (Can be skipped for this lab)**

**2.1 Min-Max Normalization**

select min(month) as min\_month, max(month) as max\_month, min(day) as min\_day, max(day) as max\_day, min(week) as min\_week, max(week) as max\_week,

min(crshour) as min\_crshour, max(crshour) as max\_crshour, min(distance) as min\_distance, max(distance) as max\_distance,

min(gap) as min\_gap, max(gap) as max\_gap, min(tmax) as min\_tmax, max(tmax) as max\_tmax, min(tmin) as min\_tmin, max(tmin) as max\_tmin,

min(prcp) as min\_prcp, max(prcp) as max\_prcp, min(snow) as min\_snow, max(snow) as max\_snow, min(awnd) as min\_awnd, max(awnd) as max\_awnd

from flight\_2007\_join;

**2.2 Feature scaling by zscore**

CREATE TABLE flight\_2007\_stddev AS

select avg(month) as avg\_month, stddev\_pop(month) as stddev\_month, avg(day) as avg\_day, stddev\_pop(day) as stddev\_day, avg(week) as avg\_week, stddev\_pop(week) as stddev\_week,

avg(crshour) as avg\_crshour, stddev\_pop(crshour) as stddev\_crshour, avg(distance) as avg\_distance, stddev\_pop(distance) as stddev\_distance,

avg(gap) as avg\_gap, stddev\_pop(gap) as stddev\_gap, avg(tmax) as avg\_tmax, stddev\_pop(tmax) as stddev\_tmax, avg(tmin) as avg\_tmin, stddev\_pop(tmin) as stddev\_tmin,

avg(prcp) as avg\_prcp, stddev\_pop(prcp) as stddev\_prcp, avg(snow) as avg\_snow, stddev\_pop(snow) as stddev\_snow, avg(awnd) as avg\_awnd, stddev\_pop(awnd) as stddev\_awnd

from flight\_2007\_join;

CREATE TABLE flight\_2007\_scaled AS

select status as label, zscore(month, avg\_month, stddev\_month) as month, zscore(day, avg\_day, stddev\_day) as day, zscore(week, avg\_week, stddev\_week) as week,

zscore(crshour, avg\_crshour, stddev\_crshour) as crshour, zscore(distance, avg\_distance, stddev\_distance) as distance,

zscore(gap, avg\_gap, stddev\_gap) as gap, zscore(tmax, avg\_tmax, stddev\_tmax) as tmax, zscore(tmin, avg\_tmin, stddev\_tmin) as tmin,

zscore(prcp, avg\_prcp, stddev\_prcp) as prcp, zscore(snow, avg\_snow, stddev\_snow) as snow, zscore(awnd, avg\_awnd, stddev\_awnd) as awnd

from flight\_2007\_join cross join flight\_2007\_stddev;

**3. Prepare the training and test dataset**

CREATE TABLE flight\_2007\_training AS

select status as label, array(month, day, week, crshour, distance, gap, tmax, tmin, prcp, snow, awnd) as features

from flight\_2007\_join;

CREATE TABLE flight\_2008\_test AS

select row\_number() over () as rowid, status as label, array(month, day, week, crshour, distance, gap, tmax, tmin, prcp, snow, awnd) as features

from flight\_2008\_join;

create table flight\_2008\_test\_exploded as

select rowid, label, feature from flight\_2008\_test LATERAL VIEW explode(features) t AS feature;

**4. Training using Logistic Regression**

create table lr\_model as

select feature, cast(avg(weight) as float) as weight

from (select logress(features, label, "-total\_steps 10000000") as (feature,weight) from flight\_2007\_training) t

group by feature;

**5. Prediction using test dataset**

create table lr\_predict as

select t.rowid, t.label, sigmoid(sum(m.weight)) as prob from flight\_2008\_test\_exploded t LEFT OUTER JOIN lr\_model m ON (t.feature = m.feature)

group by t.rowid, t.label order by rowid ASC;

**6. Generate the prediction metrics and calculate precison, recall and accuracy**

create table lr\_metrics as

select sum(case when (label = 1 and prob >= 0.5) then 1 else 0 end) as tp, sum(case when (label = 0 and prob < 0.5) then 1 else 0 end) as tn,

sum(case when (label = 1 and prob < 0.5) then 1 else 0 end) as fp, sum(case when (label = 0 and prob >= 0.5) then 1 else 0 end) as fn

from lr\_predict;

select tp/(tp+fp) as precision, tp/(tp+fn) as recall, (tp+tn)/(tp+tn+fp+fn) as accuracy from lr\_metrics;