set hive.cli.print.header=true;

create table session\_info\_non\_part (

access\_id int,

session\_name string,

session\_bytes bigint,

routing array<string>,

session\_key map<string,int>,

cust\_act\_stat struct<cust\_type:string,prod\_id:int, act\_stat:int>

)

row format delimited

fields terminated by ','

collection items terminated by '$'

map keys terminated by '#';

load data local inpath '/root/scripts/hive\_load.csv' overwrite into table session\_info\_non\_part;

###select all columns and rows

select \* from session\_info\_non\_part;

###filter on rows and columns

select \* from session\_info\_non\_part where access\_id=100;

############## partitioned tables ###########

set hive.cli.print.header=true;

create table session\_info\_part (

temp\_access\_id int,

session\_name string,

session\_bytes bigint,

routing array<string>,

session\_key map<string,int>,

cust\_act\_stat struct<cust\_type:string,prod\_id:int, act\_stat:int>

)

PARTITIONED BY (access\_id int)

row format delimited

fields terminated by ','

collection items terminated by '$'

map keys terminated by '#';

load data local inpath '/root/scripts/hive\_load.csv' overwrite into table session\_info\_part PARTITION(access\_id=100);

load data local inpath '/root/scripts/hive\_load.csv' overwrite into table session\_info\_part PARTITION(access\_id=200);

Method 1:

Load all your data in a temporary table

Insert into your partitions using insert select (with a where clause on partitioned columns)

All data transfer Data transfer mediums:

Fibre channel: 100

Wireless: 200

Backup drive/batch: 300

This is how the sample data shall look like:

100,100,aaa,37495,qa$qb$qc,aaaa#100,corp$200$10,

100,200,Bbb,49145,qa$qb$qc,bbb#200,corp$200$10,

100,300,Ccc,46977,qa$qb$qc,ccc#300,corp$200$10,

100,400,Ddd,31938,qa$qb$qc,ddd#400,corp$200$10,

100,500,eee,19450,qa$qb$qc,eee#500,corp$200$10,

100,600,fff,23066,qa$qb$qc,fff#600,corp$200$10,

100,700,ggg,19184,qa$qb$qc,ggg#700,corp$200$10,

100,800,hhh,44621,qa$qb$qc,hhh#800,corp$200$10,

100,900,iii,24033,qa$qb$qc,iii#900,corp$200$10,

100,1000,jjj,31050,qa$qb$qc,jjj#1000,corp$200$10,

200,100,aaa,37495,qa$qb$qc,aaaa#100,corp$200$10,

200,200,Bbb,49145,qa$qb$qc,bbb#200,corp$200$10,

200,300,Ccc,46977,qa$qb$qc,ccc#300,corp$200$10,

200,400,Ddd,31938,qa$qb$qc,ddd#400,corp$200$10,

200,500,eee,19450,qa$qb$qc,eee#500,corp$200$10,

200,600,fff,23066,qa$qb$qc,fff#600,corp$200$10,

200,700,ggg,19184,qa$qb$qc,ggg#700,corp$200$10,

200,800,hhh,44621,qa$qb$qc,hhh#800,corp$200$10,

200,900,iii,24033,qa$qb$qc,iii#900,corp$200$10,

200,1000,jjj,31050,qa$qb$qc,jjj#1000,corp$200$10,

############## bucketed tables ###########

set hive.cli.print.header=true;

SET hive.exec.dynamic.partition = true;

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

create table session\_info\_buck (

temp\_access\_id int,

session\_name string,

session\_bytes bigint,

routing array<string>,

session\_key map<string,int>,

cust\_act\_stat struct<cust\_type:string,prod\_id:int, act\_stat:int>

)

PARTITIONED BY (access\_id int)

CLUSTERED BY (session\_name) sorted by (session\_name) INTO 2 BUCKETS

row format delimited

fields terminated by ','

collection items terminated by '$'

map keys terminated by '#';

INSERT OVERWRITE TABLE session\_info\_buck partition(access\_id) select \* from session\_info\_part;

######################################

create TEMPORARY table session\_info\_temp (

access\_id int,

session\_name string,

session\_bytes bigint,

routing array<string>,

session\_key map<string,int>,

cust\_act\_stat struct<cust\_type:string,prod\_id:int, act\_stat:int>

)

row format delimited

fields terminated by ','

collection items terminated by '$'

map keys terminated by '#';

load data local inpath '/root/scripts/hive\_load.csv' overwrite into table session\_info\_temp;

######################################

create EXTERNAL table session\_info\_ext (

access\_id int,

session\_name string,

session\_bytes bigint,

routing array<string>,

session\_key map<string,int>,

cust\_act\_stat struct<cust\_type:string,prod\_id:int, act\_stat:int>

)

row format delimited

fields terminated by ','

collection items terminated by '$'

map keys terminated by '#'

LOCATION '/home/hdfs1313';

load data local inpath '/root/scripts/hive\_load.csv' overwrite into table session\_info\_ext;

#####################

insert into

session\_info\_non\_part

SELECT

100,

'aaa',

'200',

array("qa","qb","qc"),

map('kkk',300),

named\_struct('cust\_type','corp','prod\_id',200,'act\_stat',10);

insert into

session\_info\_part

SELECT

1100,

'aaa',

'200',

array("qa","qb","qc"),

map('kkk',300),

named\_struct('cust\_type','corp','prod\_id',200,'act\_stat',10);

############### selecting complex types

select routing from session\_info\_non\_part;

select routing[0] from session\_info\_non\_part;

select cust\_act\_stat.cust\_type from session\_info\_non\_part;

select session\_key from session\_info\_non\_part where session\_key['aaaa'] is not null;

################## views ###############

CREATE VIEW session\_info\_access\_id\_100

AS

select

\*

from

session\_info\_part

where

access\_id=100;

CREATE VIEW session\_info\_access\_id\_200

AS

select

\*

from

session\_info\_part

where

access\_id=200;

## set below or else hive won’t allow cartesian product

set hive.strict.checks.cartesian.product=false;

CREATE VIEW session\_info\_access\_id\_100\_2tables

AS

select

a.access\_id,

b.session\_bytes

from

session\_info\_part a,

session\_info\_ext b

where

a.access\_id=b.access\_id and

b.access\_id=100;

explain select

a.access\_id,

b.session\_bytes

from

session\_info\_part a,

session\_info\_non\_part b

where

a.access\_id=b.access\_id and

b.access\_id=100;

Left join = inner join + all non matching rows from table declared first

Right join = inner join + all non matching rows from table declared second

Full join = inner join + all non matching rows from both tables

select \* from session\_info\_access\_id\_100\_2tables;

############### materialized views #######

CREATE MATERIALIZED VIEW session\_info\_mat\_view\_100

AS

select

\*

from

session\_info\_part

where

access\_id=100;

load data local inpath '/root/scripts/hive\_load.csv' into table session\_info\_part PARTITION(access\_id=100);

load data local inpath '/root/scripts/hive\_load.csv' into table session\_info\_part PARTITION(access\_id=200);

############### Archiving

set hive.archive.enabled=true;

set hive.archive.har.parentdir.settable=true;

set har.partfile.size=1099511627776;

set hive.metastore.schema.verification=true;

#### ignore below line #######

hive --auxpath $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-archives-2.9.2.jar

#### ignore below line #######

ALTER TABLE session\_info\_part ARCHIVE partition (access\_id=100);

#########

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;

############### UDF

set hive.cli.print.header=true;

create table hive\_udf\_example (

session\_name string

)

row format delimited

fields terminated by ','

collection items terminated by '$'

map keys terminated by '#';

insert into hive\_udf\_example values ('session1');

insert into hive\_udf\_example values ('session2');

insert into hive\_udf\_example values ('session3');

insert into hive\_udf\_example values ('session4');

—create python script, save as python\_udf\_hive.py and change permission to 755

#!/usr/bin/python

## convert string to hex

import sys

random\_salt='jkfgldfjghldfh'

for line in sys.stdin:

#print(line)

line = line.strip('\n\r')

#base16INT = int(line, 16)

#print("value", base16INT)

#print("value", type(base16INT))

hex\_value = (line + random\_salt)

#print(hex\_value)

print (hex\_value)

— add python file to hive

add file /root/scripts/python\_udf\_hive.py

—run sql

select

TRANSFORM(session\_name)

USING 'python /root/src/python\_udf\_hive.py'

from

hive\_udf\_example;

############### udf

DROP TEMPORARY MACRO IF EXISTS usage\_charge;

CREATE TEMPORARY MACRO usage\_charge(session\_bytes string)

CASE

WHEN CAST(session\_bytes as INT) > 10000 and CAST(session\_bytes as INT) <= 20000 THEN 100

WHEN session\_bytes > 20000 and session\_bytes <= 30000 THEN 200

WHEN session\_bytes > 30000 and session\_bytes <= 40000 THEN 300

WHEN session\_bytes > 40000 and session\_bytes <= 50000 THEN 400

ELSE 500

END

;

SELECT session\_bytes, usage\_charge(session\_bytes) as bill from session\_info\_non\_part;

############## udf/macro assignment

Using a Macro/UDF, find the length of every session\_name string and the bill of the user.

Bill is to be calculated as follows:

WHEN session\_bytes < 10000 THEN 50

WHEN session\_bytes > 10000 and session\_bytes <= 20000 THEN 100

WHEN session\_bytes > 20000 and session\_bytes <= 30000 THEN 200

WHEN session\_bytes > 30000 and session\_bytes <= 40000 THEN 300

WHEN session\_bytes > 40000 and session\_bytes <= 50000 THEN 400

WHEN session\_bytes > 50000 THEN 900

Output shall be: session\_id, session\_name, session\_name\_length, usage\_bill

The output needs to be used for interim processing of data further by some downstream processes. Stage the data accordingly.

############### assignment

• Modify table definition for session\_info\_non\_part such that named\_struct cust\_act\_stat also contains a map, and an array

• Modify source file to it can be used in load statement to load this table

• Modify insert statement with corresponding dummy values so it can insert into this table

• Try the same exercise with different separators (column, collection, key)