Retail Customer Segmentation Data lake and Analytics

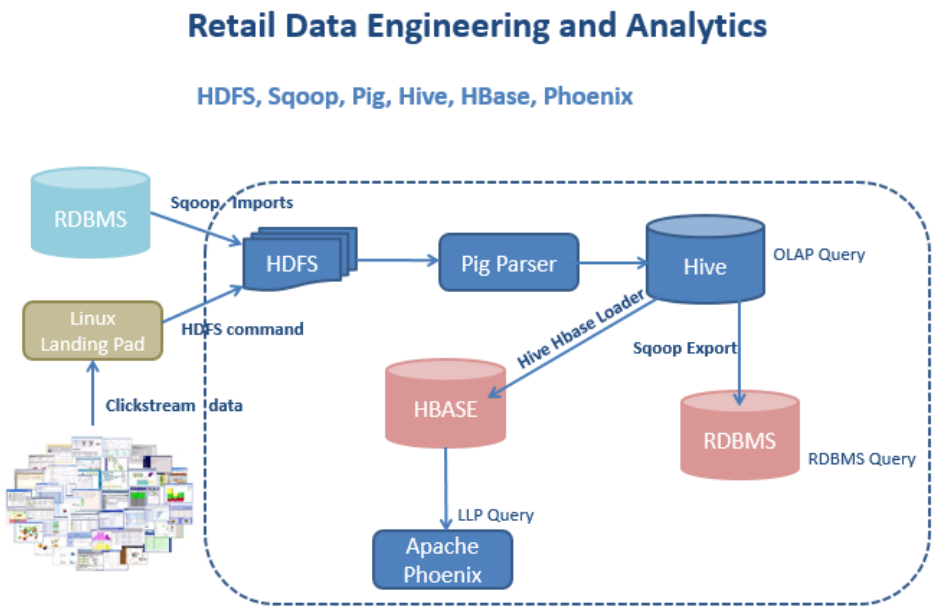
Important Note (Please read) :

If you spend your valuable time in executing this project reading each and every line of comments and rather than just copy paste the code and execute, for sure you can get a complete understanding of the realworld project with comprehensive knowledge in HDFS, Sqoop, Pig, Hive, Hbase and Phoenix.

Project Synopsis:

Building of this Data lake project provides an all in one central repository for converged Data Platform that helps retailers chain of stores to integrate and analyze a wide variety of online and offline customer data including the customer data, products, purchases, orders, employees and comments data. Retailers can analyze this data to generate insights about individual consumer behaviors and preferences, Recommendations and Loyalty Programs. This tool builds the strategies, Key Performance Indicators (KPI) definitions and implementation roadmaps that assists our esteemed clients in their Analytics & Information ecosystem journey right from Strategy definition to large scale Global Implementations & Support using the bigdata ecosystems.

Architecture:



Project Flow:

1. **Sqoop** for injecting the data from different databases/schemas using most of the options.
2. **HDFS** for persisting the data for primary staging.
3. **Pig** to parse
    a. Convert to complex data type, reduce/change the order of columns extracted from the database, Filter the custdetails complex data which has non zero amounts.

   b. Split based on the amount of the product, reorder the columns, union and store the output into HDFS.
4. **Hive** to create and load ..
   a. Under staging Database create **managed** temporary tables, Load the Pig output data into the above managed tables that will be dropped and recreated on daily basis.
   b. Under retail mart database create and load **external tables** which is **partitioned** for applying where clauses and **bucketed** to join with the other staging tables.
   c. Index the high cardinal values.
   d. Create a final external table which is **partitioned** and not bucketed.
   e. Customer Website viewship pattern and Frustration Scoring use case queries.
   f. Create a hive – hbase handler table to load the refined data into hbase.
5. **Sqoop Export** to load the aggregated data to RDBMS again .
6. **HBase** to view the data generated in the hive handler table.
7. **Phoenix** to create a table on top of hbase table to perform low latency queries.

Prerequisites:

Ensure – Hadoop, Sqoop, Hive, Pig, HBase and Phoenix are installed in the node.

Login to VMWare –

Legend:

Blue Color – Commands/SQLs/Scripts

Black color with bold and underline – Represents headings

Plain black/red text – Represents descriptions

Brown text – Represents some additional options/usecases.

Go to:

Player -> Manage -> Virtual Machine Settings -> change the memory as 3072 MB and cpu core to 2

Preparation of Source data: Untar the data provided in retailorders.tar.gz

```
cd ~
tar xvzf retailorders.tar.gz
```

Start the Following Eco systems

Start Hadoop:

```
start-all.sh
mr-jobhistory-daemon.sh start historyserver
```

Login, start mysql service and exit:

```
sudo su mysql
password:hduser
```

service mysqld start

exit

MYSQL - Preparing the Source DB data ready:

Login to mysql using root user:

mysql -u root -p

password: root

mysql

Run the below sqls to create and load data in the tables created in the below schemas

ordersproduct_ORIG.sql - Schema: ordersproducts, Tables: orders, products, orderdetails

custpayments_ORIG.sql - Schema: custpayments, Tables: custpayments and paymentspayments

empoffice.sql - Schema: empoffice, Tables: employees and offices

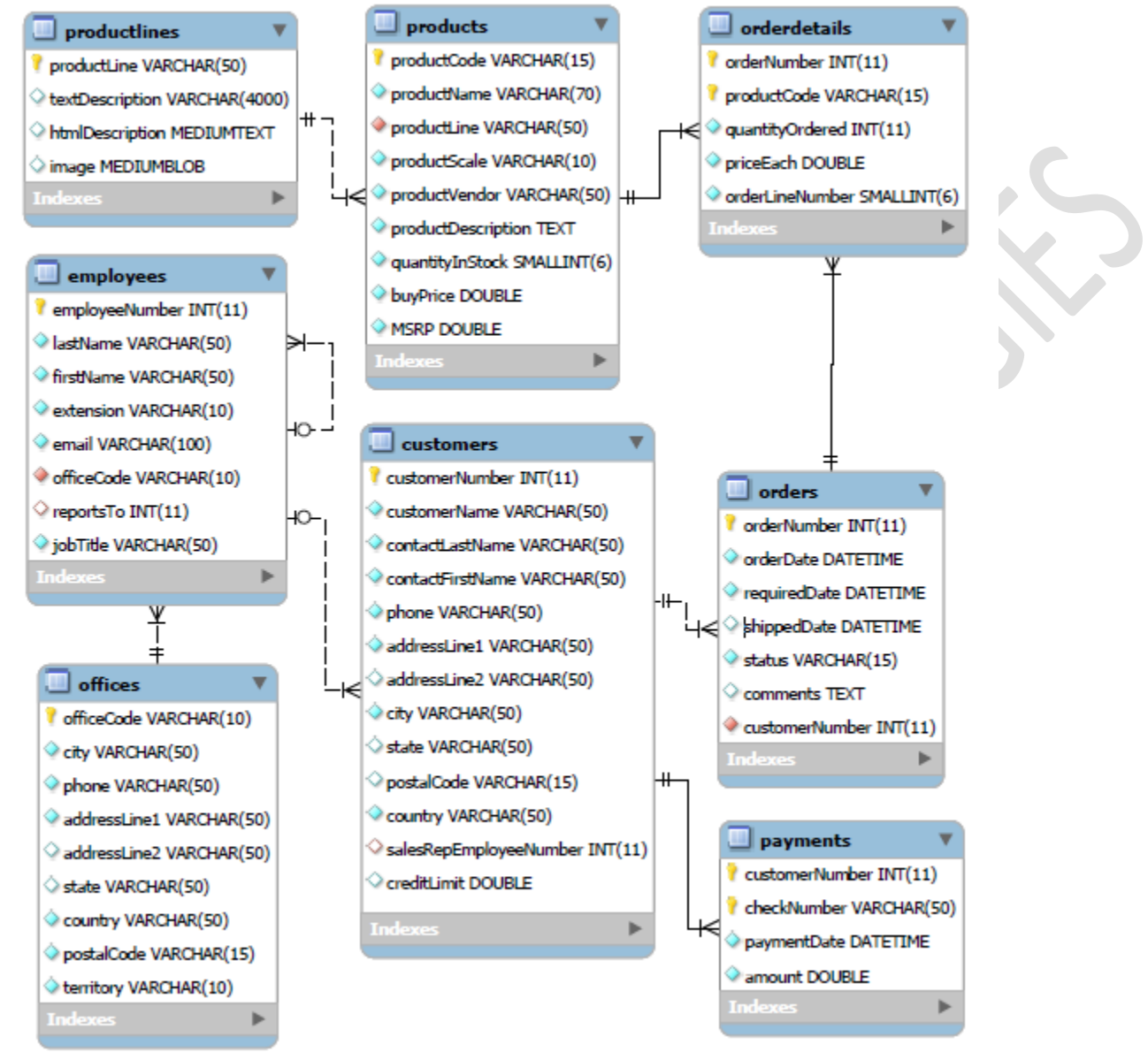
mysql>

source /home/hduser/retailorders/ordersproduct_ORIG.sql

source /home/hduser/retailorders/custpayments_ORIG.sql

source /home/hduser/retailorders/empoffice.sql

Table Schema:



SQOOP:

Create a password file adding root as password:

```
echo -n "root" > ~/root.password
```

Create the below directories in hdfs and place the password file

```
hadoop fs -mkdir /user/hduser/retailorders/

hadoop fs -put ~/root.password /user/hduser/retailorders/root.password

hadoop dfs -chown 400 /user/hduser/retailorders/root.password
```

Run the below Sqoop import commands with options file, password file, boundary query, query, split by, delete target directory, null string, direct mode, ~ delimiter, reading without splitby column with primary key as split by

Import the employees table from the DB

```
sqoop --options-file /home/hduser/retailorders/empofficeoption --password-file
/user/hduser/retailorders/root.password -table employees -m 2 --delete-target-dir --target-dir employees/ --
fields-terminated-by '~' --lines-terminated-by '\n';
```

Import the offices table from the DB

```
sqoop --options-file /home/hduser/retailorders/empofficeoption --password-file
/user/hduser/retailorders/root.password -table offices -m 1 --delete-target-dir --target-dir offices/  --fields-
terminated-by '~' --lines-terminated-by '\n';
```

Import the Customer and payments data joined with most of the options used for better optimization and best practices

```
sqoop --options-file /home/hduser/retailorders/custoption --password-file
/user/hduser/retailorders/root.password --boundary-query " select min(customerNumber),
max(customerNumber) from payments " --query 'select c.customerNumber,
upper(c.customerName),c.contactFirstName,c.contactLastName,c.phone,c.addressLine1,c.city,c.state,c.postalCo
de,c.country ,c.salesRepEmployeeNumber,c.creditLimit ,p.checknumber,p.paymentdate,p.amount
from customers c inner join payments p on c.customernumber=p.customernumber
where $CONDITIONS' \
--split-by c.customernumber --delete-target-dir --target-dir custdetails/2016-10/ --null-string 'NA' \
--direct --num-mappers 2 --fields-terminated-by '~' --lines-terminated-by '\n';
```

Import the orders, orderdetail and products data joined with most of the options used for better optimization and best practices

```
sqoop --options-file /home/hduser/retailorders/ordersoption --password-file
/user/hduser/retailorders/root.password --boundary-query "select min(customerNumber),
max(customerNumber) from orders" --query 'select
o.customernumber,o.ordernumber,o.orderdate,o.shippeddate,o.status,o.comments,od.productcode,od.quantity
ordered,od.priceeach,
od.orderlinenumber,p.productCode,p.productName,p.productLine,p.productScale,p.productVendor,p.productDe
scription,p.quantityInStock,p.buyPrice,p.MSRP
from orders o inner join orderdetails od on o.ordernumber=od.ordernumber
inner join products p on od.productCode=p.productCode where $CONDITIONS' \
--split-by o.customernumber --delete-target-dir --target-dir orderdetails/2016-10/ --null-string 'NA' \
--direct --num-mappers 4 --fields-terminated-by '~' --lines-terminated-by '\n';
```

PIG

1. Login to Pig
2. Load the cust and order details data data from the above sqoop imported location.
3. Convert the custdetails data into complex data types (struct in hive) and store in a hdfs location.
4. Read the order details data into pig relation to reduce/change the order of columns extracted from the database.
5. Store the output into pigout/custdetcomplextypes and pigout/orddetails/ locations in HDFS.

```
pig -x mapreduce

custdetails = load '/user/hduser/custdetails/2016-10' USING PigStorage ('~') as
(customerNumber:chararray,customerName:chararray,contactFirstName:chararray,contactLastName:chararray,p
hone:chararray,addressLine1:chararray,city:chararray,state:chararray,postalCode:chararray,counrtry:chararray,sa
lesRepEmployeeNumber:chararray,creditLimit:chararray,checknumber:chararray,paymentdate:chararray,amount
:chararray) ;

custdetcomplextypes = foreach custdetails generate
CONCAT(customerNumber,'~',customerName,'~',CONCAT(contactFirstName,'
',contactLastName),'~',CONCAT(addressLine1,'$',city,'$',state,'$',postalCode,'$',counrtry,'$',phone),'~',creditLimit,
'~',checknumber,'~',amount,'~',paymentdate);

store custdetcomplextypes into '/user/hduser/pigout/custdetcomplextypes' using PigStorage ('~');

orderdetails = load '/user/hduser/orderdetails/2016-10' USING PigStorage ('~') as
(customernumber:chararray,ordernumber:chararray,orderdate:chararray,shippeddate:chararray,status:chararray
,comments:chararray,productcode:chararray,quantityordered:chararray,priceeach:chararray,orderlinenumber:ch
ararray,productCode:chararray,productName:chararray,productLine:chararray,productScale:chararray,productVe
ndor:chararray,productDescription:chararray,quantityInStock:chararray,
buyPrice:chararray,MSRP:chararray) ;
```

```
orddetcomplextypes = foreach orderdetails generate
customernumber,ordernumber,shippeddate,status,comments,productcode,quantityordered,priceeach,orderline
number,productName,productLine,
productScale,productVendor,productDescription,quantityInStock,buyPrice,MSRP,orderdate;

store orddetcomplextypes into '/user/hduser/pigout/orddetails/' using PigStorage ('~');
```

Below steps 5, 6 and 7 are additional usecases/iterations

5. load and convert the custdetails data to array data and Store into HDFS location.

```
custpaymentcomplextypes = foreach custdetails generate
CONCAT(customerNumber,'~',checknumber,'~',CONCAT(creditLimit,'$',amount),'~',paymentdate);

store custpaymentcomplextypes into '/user/hduser/pigout/custpaymentcomplextypes' using PigStorage ('~');
```

6. Filter the custdetails complex data which has non zero amounts.
7. Split based on the amount of the product, reorder the columns, union and store the output into HDFS.

```
custfiltered = filter custdetails by (int)amount > 0;

SPLIT custfiltered INTO highamt IF ( (int)amount>50000 ) , midamt IF ( (int)amount <50000 AND (int)amount >
10000), lowamt IF ((int)amount<10000);

lamt = foreach lowamt generate $0,$12,$14,'lowamt',$13;
mamt = foreach midamt generate $0,$12,$14,'midamt',$13;
hamt = foreach highamt generate $0,$12,$14,'highamt',$13;
allamt = UNION lamt,mamt,hamt;

store allamt into '/user/hduser/pigout/allamt' using PigStorage ('~');
```

HIVE:

Start Hive metastore in a terminal:

```
hive --service metastore
```

Start hive cli in another terminal:

```
hive
```

1. Managed Table use case: Under **staging** retail_stg Database **create temporary managed tables**, Load the Pig output data into the below managed tables **that will be dropped and recreated on daily basis so data will be cleaned when dropped, if we use external table we have to manage the data cleanup separately.**

```
drop database if exists retail_stg cascade;
```

```
create database retail_stg;
use retail_stg; drop table if exists custdetstg;

create table custdetstg(customernumber STRING, customername STRING, contactfullname string, address
struct<addressLine1:string,city:string,state:string,postalCode:bigint,counrtry:string,phone:bigint>, creditlimit
float,checknum string,checkamt int,paymentdate date)
row format delimited
fields terminated by '~'
collection items terminated by '$'
stored as textfile;

load data inpath '/user/hduser/pigout/custdetcomplextypes' overwrite into table custdetstg;

drop table if exists orddetstg;

create table orddetstg(customernumber STRING, ordernumber STRING, shippeddate date,status string,
comments string,productcode string,quantityordered int,priceeach decimal(10,2),orderlinenumber
int,productName STRING,productLine STRING,
productScale STRING,productVendor STRING,productDescription STRING,quantityInStock int,buyPrice
decimal(10,2),MSRP decimal(10,2),orderdate date)
row format delimited
fields terminated by '~'
stored as textfile;

load data inpath '/user/hduser/pigout/orddetails/' overwrite into table orddetstg;
```

(or)

To run the above statements one after another by running outside hive cli ie from the linux terminal:

$ hive -e "use retail_stg; drop table if exists custdetstg;"

(or)

Run the above hive queries as a batch hql job: create a hql file and copy all the above lines and run the hql script.
Goto pig grunt shell and re execute the below load commands alone to recreate the output files that are used as input to the hive tables:

```
grunt> store custdetcomplextypes into '/user/hduser/pigout/custdetcomplextypes' using PigStorage ('~');
grunt> store orddetcomplextypes into '/user/hduser/pigout/orddetails/' using PigStorage ('~');

cd ~/retailorders

vi staging_tbls.hql
use retail_stg;

drop table if exists custdetstg;

create table custdetstg(customernumber STRING, customername STRING, contactfullname string, address struct<addressLine1:string,city:string,state:string,postalCode:bigint,counrtry:string,phone:bigint>, creditlimit float,checknum string,checkamt int,paymentdate date)
row format delimited
fields terminated by '~'
```

```
collection items terminated by '$'
stored as textfile;

load data inpath '/user/hduser/pigout/custdetcomplextypes' overwrite into table custdetstg;

drop table if exists orddetstg;

create table orddetstg(customernumber STRING, ordernumber STRING, shippeddate date,status string, comments string,productcode string,quantityordered int,priceeach decimal(10,2),orderlinenumber int,productName STRING,productLine STRING,
productScale STRING,productVendor STRING,productDescription STRING,quantityInStock int,buyPrice decimal(10,2),MSRP decimal(10,2),orderdate date)
row format delimited
fields terminated by '~'
stored as textfile;

load data inpath '/user/hduser/pigout/orddetails/' overwrite into table orddetstg;
```

hive -f /home/hduser/retailorders/staging_tbls.hql

2. External tables with partition/bucketing usecases : Under retail_mart database create and load external tables which is partitioned for applying where clauses and bucketed to join with the other external tables more efficiently. **Why we are creating as external table because we may keep on adding more data in this table or performing more iterative queries without dropping it as we have to join these tables to produce the final mart.**

```
hadoop fs -rmr /user/hduser/custorders/custdetpartbuckext

drop database if exists retail_mart cascade;
create database retail_mart;
use retail_mart;

set hive.enforce.bucketing = true ;
set map.reduce.tasks = 3;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.dynamic.partition=true;

create external table custdetpartbuckext (customernumber STRING, customername STRING, contactfullname
string, address
struct<addressLine1:string,city:string,state:string,postalCode:bigint,counrtry:string,phone:bigint>,creditlimit
float,checknum string,checkamt int)
partitioned by (paymentdate date)
clustered by (customernumber) INTO 3 buckets
row format delimited
fields terminated by '~'
collection items terminated by '$'
stored as textfile
location '/user/hduser/custorders/custdetpartbuckext';

insert into table custdetpartbuckext partition(paymentdate) select customernumber,customername,
contactfullname, address ,creditlimit,checknum,checkamt,paymentdate from retail_stg.custdetstg ;

create external table orddetpartbuckext(customernumber STRING, ordernumber STRING, shippeddate
date,status string, comments string,productcode string,quantityordered int,priceeach
decimal(10,2),orderlinenumber int,productName STRING,productLine STRING,
productScale STRING,productVendor STRING,productDescription STRING,quantityInStock int,buyPrice
decimal(10,2),MSRP decimal(10,2))
partitioned by (orderdate date)
clustered by (customernumber) INTO 3 buckets
```

```
row format delimited
fields terminated by '~'
collection items terminated by '$'
stored as textfile
location '/user/hduser/custorders/orddetpartbuckext';

insert into table orddetpartbuckext partition(orderdate) select customernumber,ordernumber,
shippeddate,status,comments,productcode,quantityordered ,priceeach ,orderlinenumber ,productName
,productLine,
productScale,productVendor,productDescription,quantityInStock,buyPrice,MSRP,orderdate from
retail_stg.orddetstg ;
```

3. Create a final external tables with orc and text format because it saves space and query will be executed much faster, and why we are loading this final table with orc format is due to intermediate tables like staging doesn't require serialized data and is partitioned because we may apply date filter for querying this table and not bucketed because we will not joining this final table with any other table.

4. Index the high cardinal values.

The below table marked in red is a orc table without partition and the later table marked in blue is partitioned table without orc, do a benchmark after creating and loading tables running queries.

```
create external table custordfinal (
customernumber STRING, customername STRING, contactfullname string, addressLine1 string,city string,state
string,country string,phone bigint,creditlimit float,checknum string,checkamt int,ordernumber STRING,
shippeddate date,status string, comments string,productcode string,quantityordered int,priceeach
decimal(10,2),orderlinenumber int,productName STRING,productLine STRING,productScale
STRING,productVendor STRING,productDescription STRING,quantityInStock int,buyPrice decimal(10,2),MSRP
decimal(10,2),orderdate date)
row format delimited
fields terminated by '~'
stored as orcfile
location '/user/hduser/custorders/custordfinal';

insert into table custordfinal
select cd.customernumber,cd.customername, cd.contactfullname,
cd.address.addressLine1,cd.address.city,cd.address.state,cd.address.counrtry,cd.address.phone
,cd.creditlimit,cd.checknum,cd.checkamt,
o.ordernumber, o.shippeddate,o.status,o.comments,o.productcode,o.quantityordered ,o.priceeach
,o.orderlinenumber ,o.productName ,o.productLine,
productScale,o.productVendor,o.productDescription,o.quantityInStock,o.buyPrice,o.MSRP,o.orderdate
from custdetpartbuckext cd inner join orddetpartbuckext o on cd.customernumber=o.customernumber ;

create external table custordpartfinal (
customernumber STRING, customername STRING, contactfullname string, addressLine1 string,city string,state
string,country string,phone bigint,creditlimit float,checknum string,checkamt int,ordernumber STRING,
shippeddate date,status string, comments string,productcode string,quantityordered int,priceeach
decimal(10,2),orderlinenumber int,productName STRING,productLine STRING,productScale
```

```
STRING,productVendor STRING,productDescription STRING,quantityInStock int,buyPrice decimal(10,2),MSRP
decimal(10,2),orderdate date)
partitioned by (paymentdate date)
row format delimited
fields terminated by '~'
stored as textfile
location '/user/hduser/custorders/custordpartfinal';

create index idx_custordpartfinal_phone on table custordpartfinal(phone) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;

insert into table custordpartfinal partition(paymentdate)
select cd.customernumber,cd.customername, cd.contactfullname,
cd.address.addressLine1,cd.address.city,cd.address.state,cd.address.counrtry,cd.address.phone
,cd.creditlimit,cd.checknum,cd.checkamt,
o.ordernumber, o.shippeddate,o.status,o.comments,o.productcode,o.quantityordered ,o.priceeach
,o.orderlinenumber ,o.productName ,o.productLine,
productScale,o.productVendor,o.productDescription,o.quantityInStock,o.buyPrice,o.MSRP,o.orderdate,cd.payme
ntdate
from custdetpartbuckext cd inner join orddetpartbuckext o on cd.customernumber=o.customernumber ;
```

Customer Website viewship pattern and Frustration Scoring use case (Important)

Below usecase will be used for exploring (UDTF) User defined tabular function works on one row as input and returns multiple rows as output. So here the relation in one to many. e.g Hive built in EXPLODE() function. UDTF can be used to split a column into multiple column as well which we will look in below example. Here alias "AS" clause is mandatory . This usecase also explains the use of analytical functions like row_number, rank_over etc.

1. Create the order_rate table in staging and load the static data hold info about comments-keywords, company or customer comments, siverity value.

```
use retail_stg;
drop table if exists order_rate;

create table retail_stg.order_rate (rid int,orddesc varchar(200),comp_cust varchar(10),siverity int) row format
delimited fields terminated by ',';

load data local inpath '/home/hduser/retailorders/orders_rate.csv' overwrite into table order_rate;
```

2. Create the orddetstg table in staging and load the stgdata holds info about customernumber, comments given by the company about the customer, pagenavigated by the customers and the navigation order. We are going to apply UDTF to pivot the array data into multiple rows to easily navigate and relate the page viewship of the customers.

```
drop table if exists orddetstg;
```

```
create table  retail_stg.orddetstg (customernumber string,comments string,pagenavigation array
<string>,pagenavigationidx array <int>) row format delimited fields terminated by ','
collection items terminated by '$';

load data local inpath '/home/hduser/retailorders/stgdata' overwrite into table retail_stg.orddetstg;
hadoop fs -rmr /user/hduser/custmart/

drop table if exists retail_mart.cust_navigation;

create external table  retail_mart.cust_navigation (customernumber string,navigation_pg string,navigation_index
int) row format delimited fields terminated by ','
location '/user/hduser/custmart/';
```

3. Below insert query loads the **pivoted** data using posexplode **UTAF** function which converts the array elements for eg. From custid: 1, pagenavigation array [homescreen$advertisement$purchasescreen$exit] to the data given below.

custid pagenavigation
1 homescreen
1 advertisement
1 purchasescreen
1 exit

```
insert overwrite table retail_mart.cust_navigation select customernumber,pgnavigation as pagenavig
,pgnavigationidx as pagenavigindex
from retail_stg.orddetstg
lateral view posexplode(pagenavigation) exploded_data1 as x, pgnavigation
lateral view posexplode(pagenavigationidx) exploded_data2 as y,  pgnavigationidx
where x=y;
```

4. Below select query using the analytical functions provides the **reverse** ordering of the page navigation from higher value to low like given below.

Custid pagenavigation navigationindex reversed
496 exit 1
496 order 2
496 cart 3
496 profile 4
496 about-us 5
496 home 6

```
select customernumber,navigation_pg ,row_number() over (partition by customernumber order by
navigation_index desc) as visit_number from retail_mart.cust_navigation;
```

5. Frustration value of the customers are identified by joining the static table order_rate created in the first step with the retail_stg.orddetstg table created later to match with the pattern and aggregate the frustration score then find the frustration level.

```
create external table  retail_mart.cust_frustration_level (customernumber string,total_siverity
int,frustration_level string) row format delimited fields terminated by ','
location '/user/hduser/custmartfrustration/';

insert overwrite table retail_mart.cust_frustration_level
select customernumber,total_siverity,case when total_siverity between -10 and -3 then 'highly frustrated' when
total_siverity between -2 and -1 then 'low frustrated'
when total_siverity = 0 then 'neutral' when total_siverity between 1 and 2 then 'happy' when total_siverity
between 3 and 10 then 'overwhelming' else 'unknown' end as customer_frustration_level from (
select customernumber,sum(siverity) as total_siverity from (
select o.customernumber,o.comments,r.orddesc,siverity from retail_stg.orddetstg o left outer join order_rate r
where o.comments like concat('%',r.orddesc,'%')) temp1
group by customernumber) temp2;
```

```
+---------------+---------------+------------------+
| customernumber | total_siverity | frustration_level |
+---------------+---------------+------------------+
| 201           |            -5 | highly frustrated |
| 202           |             2 | happy             |
| 205           |            -2 | low frustrated    |
| 216           |             5 | overwhelming      |
| 242           |             5 | overwhelming      |
| 362           |             3 | overwhelming      |
| 452           |            -5 | highly frustrated |
| 456           |             5 | overwhelming      |
| 496           |            -2 | low frustrated    |
| 112           |            -4 | highly frustrated |
| 124           |             3 | overwhelming      |
| 128           |            -4 | highly frustrated |
```

5. **Sqoop** **Export to load the aggregated data to RDBMS again for frontend legacy reports, the above processing couldn't be done in the RDBMS environment , hence off loaded to hadoop platform and did all above processing and final aggregated data is brought back to DB**.

```
mysql -u root -p
password: root

create database customer_reports;

CREATE TABLE customer_reports.customer_frustration_level ( customernumber varchar(200), total_siverity
float,frustration_level varchar(100) );

sqoop export --connect jdbc:mysql://localhost/customer_reports --username root --password root --table
customer_frustration_level --export-dir /user/hduser/custmartfrustration/
```

Hive – Hbase Integeration:

6. Linux - Copy all hbase lib files to hive lib directories and add the below env variable:

```
cp /usr/local/hbase/lib/hbase-common-0.98.4-hadoop2.jar /usr/local/hive/lib/
cp /usr/local/hbase/lib/zookeeper-3.4.6.jar /usr/local/hive/lib/
cp /usr/local/hbase/lib/guava-12.0.1.jar /usr/local/hive/lib/
cp /usr/local/hbase/lib/hbase-protocol-0.98.4-hadoop2.jar /usr/local/hive/lib/
cp /usr/local/hbase/lib/hbase-server-0.98.4-hadoop2.jar /usr/local/hive/lib/

cd /usr/local/hive/conf/

mv hive-env.sh.template hive-env.sh

vi hive-env.sh
export HIVE_AUX_JARS_PATH=/usr/local/hbase/lib
```

Ensure zookeeper and hbase are started:

```
zkServer.sh start
start-hbase.sh
```

7. Start Hive cli invoking hbase libraries to connect to hbase and zookeeper with hive.

```
hive --auxpath /usr/local/hive/lib/hive-hbase-handler-0.14.0.jar , /usr/local/hive/lib/hbase-common-0.98.4-hadoop2.jar , /usr/local/hive/lib/zookeeper-3.4.6.jar , /usr/local/hive/lib/guava-12.0.1.jar ,/usr/local/hive/lib/hbase-protocol-0.98.4-hadoop2.jar , /usr/local/hive/lib/hbase-server-0.98.4-hadoop2.jar -hiveconf hbase hbase.master=masternode:60000 hive.root.logger=INFO,console hbase.zookeeper.quorum=localhost:2181
```

8. Create the **hive hbase handler table** and load the data into hive table by insert select, which in turns loads to hbase table mapping the column family and columns. Here reflect("java.util.UUID", "randomUUID") is used to create sequence number.

```
use retail_mart;

CREATE TABLE custordprodtbl (key varchar(100),customernumber varchar(100),productLine varchar(100),state varchar(100),city varchar(100),creditlimit float,checknum varchar(200),checkamt float,ordernumber varchar(200),status varchar(100),
comments varchar(4000),productcode varchar(100),quantityordered int,priceeach decimal(10,2),productName varchar(1000),buyPrice decimal(10,2),MSRP decimal(10,2),orderdate date,paymentdate date)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
```

```
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,cust:custno,prod:prodline,cust:state,cust:city,cust:creditlimit,cust:checknum,cust:checkamt,ord:ordnum,ord:status,ord:comments,prod:prodcd,prod:qty,prod:price,prod:prodname,prod:buyprice,prod:msrp,ord:orddt,cust:paydt")
TBLPROPERTIES ("hbase.table.name" = "custordprodtbl", "hbase.mapred.output.outputtable"
="custordprodtbl");

insert into table custordprodtbl SELECT reflect("java.util.UUID", "randomUUID") as
key,customernumber,productLine,state,city,creditlimit,checknum,checkamt,ordernumber,
status,comments,productcode,quantityordered ,priceeach ,productName,
buyPrice,MSRP,orderdate,paymentdate
from custordpartfinal;
```

9. **Login to HBase** to view the data generated in the hive handler table.

```
hbase shell
scan 'custordprodtbl'
```

10. **Phoenix** to create a table on top of hbase table to perform low latency queries.

```
stty cols 200
sqlline.py localhost

!set maxwidth 1000
create TABLE "custordprodtbl" ("key" varchar(100) PRIMARY KEY,"cust"."custno" varchar(100),"prod"."prodline"
varchar(100),"cust"."state" varchar(100),"cust"."city" varchar(100),"cust"."creditlimit" float,"cust"."checknum"
varchar(200),"cust"."checkamt" float,"ord"."ordnum" varchar(200),"ord"."status"
varchar(100),"ord"."comments" varchar(4000),"prod"."prodcd" varchar(100),"prod"."qty" integer,"prod"."price"
float,"prod"."prodname" varchar(1000),"prod"."buyprice" float,"prod"."msrp" float,"ord"."orddt"
varchar(10),"cust"."paydt" varchar(10));

select "cust"."custno","prod"."prodline","cust"."state","cust"."city","cust"."creditlimit","cust"."checkamt" from
"custordprodtbl" WHERE "ord"."comments" like 'Custom%';
```

Thank You!!

This project has a good scope to enhance adding pig - hbase integration, hcatalog hive - pig integration and hive - elastic search - kibana integration.