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 of Building of Data Lake with comprehensive knowledge in HDFS, Sqoop, Hive, Hbase and Phoenix.

Project Synopsis:

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.

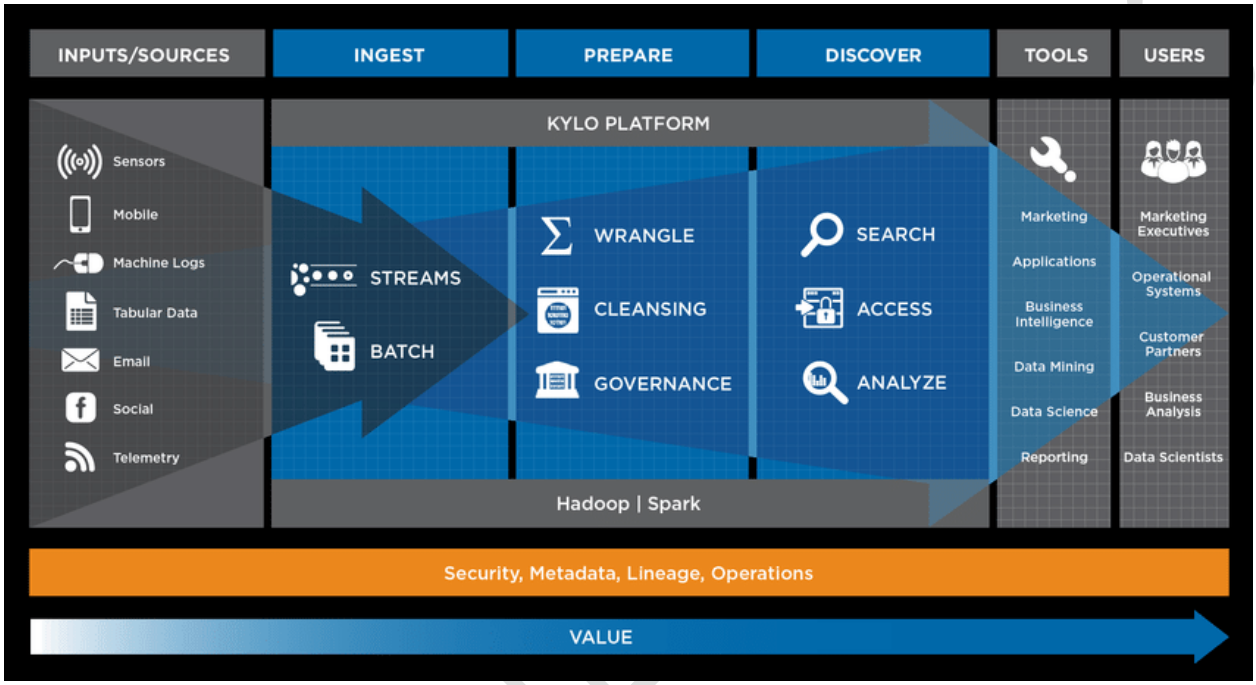
The successful BigData journey typically starts with new analytic applications, which lead to a Data Lake. As more and more applications are created that derive value from the new types of data from sensors/machines, server logs, clickstreams and other sources, the Data Lake forms with Hadoop acting as a shared service for delivering deep insight across a large, broad, diverse set of data at efficient scale in a way that existing enterprise systems and tools can integrate with and complement the Data Lake journey.

As most of you know, Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs. The Benefits of Hadoop is the computing power, flexibility, low cost, horizontally scalable , fault tolerant and lots more..

There are multiple options which are used in the industry to move the data from Legacy Systems to the Hadoop Ecosystems. We can how use the capabilities of Hortonworks Data Platform to perform the Cleansing, Aggregation, Transformation, Machine Learning, Search, Visualizing etc

Why Datalake

- ✓ Unify data from more sources and formats
- ✓ Federate and query virtually any data
- ✓ Drive machine learning and advanced analytics
- ✓ Streamline data preparation and access
- ✓ Reduce IT and warehouse costs
- ✓ Improve data-driven decisions



Zones of Data Lake:

Transient Zone :

*Used to hold ephemeral **data**, such as temporary copies, streaming spools, or other short-lived **data** before being ingested. Mostly file systems such as HDFS or Linux landing pad will be used.*

Raw Zone:

Raw Zone will contain attributes and columns from sources that will not entirely make the journey to the Curated or Consumer Zones. The benefit of a Raw Zone is that it allows new sources to be ingested quickly and the lineage of the data in Zones further along in processing to be established. Mostly file systems such as HDFS or Hive will be used for raw zone.

Curated Zone:

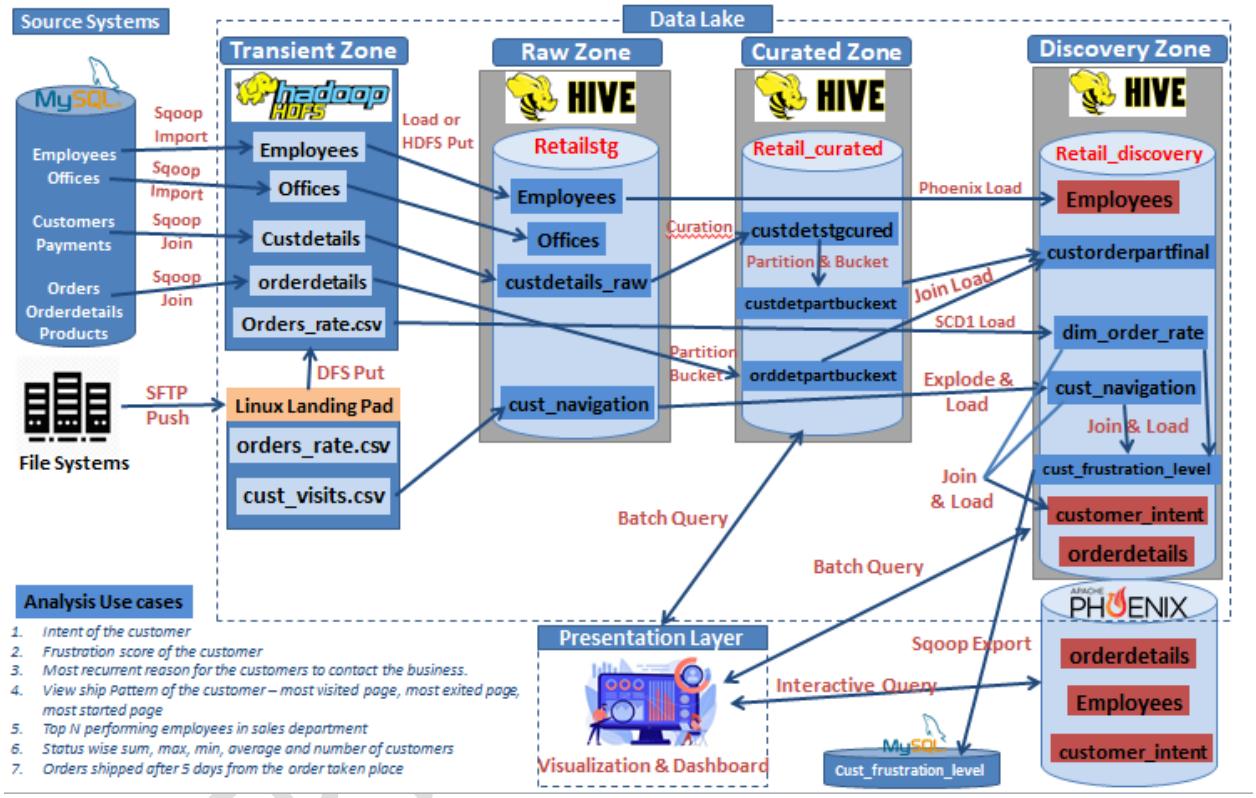
Curated data that is often stored in a data model, which combines like data from a variety of sources (often referred to as a canonical model). This Zone may be used to feed an external data warehouse or serve as the organization's data warehouse. The data from one source may be augmented with calculations, aggregations, or data from another source in this zone. Checks for referential integrity,

data quality or missing data can be performed in this zone. Hive or any other NOSQLs will be used for curated zones.

Discovery or Consumption Zone:

While the Curated Zone can be thought of as the data warehouse of the data lake, the Consumer Zone can be thought of as the data marts for the lake. The data in this zone is organized so that consumers can easily fetch what they need for their analysis, reporting, derivation of metrics and KPIs. Feeds to downstream systems can be created from this zone of the lake.

Project Tech Stack Architecture and Data Flow Diagram:



Tech Stacks Used:

1. **Sqoop** for injecting the data from different databases/schemas using most of the options.
2. **HDFS/LFS** for persisting the data for primary staging.
3. **Hive** 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 Sqoop imported 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 – Phoenix handler table to load the refined data into hbase.

5. **Sqoop Export** to load the aggregated data to RDBMS again .
6. **Phoenix** to perform low latency queries.

Prerequisites:

Ensure – Hadoop, Sqoop, Hive, 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

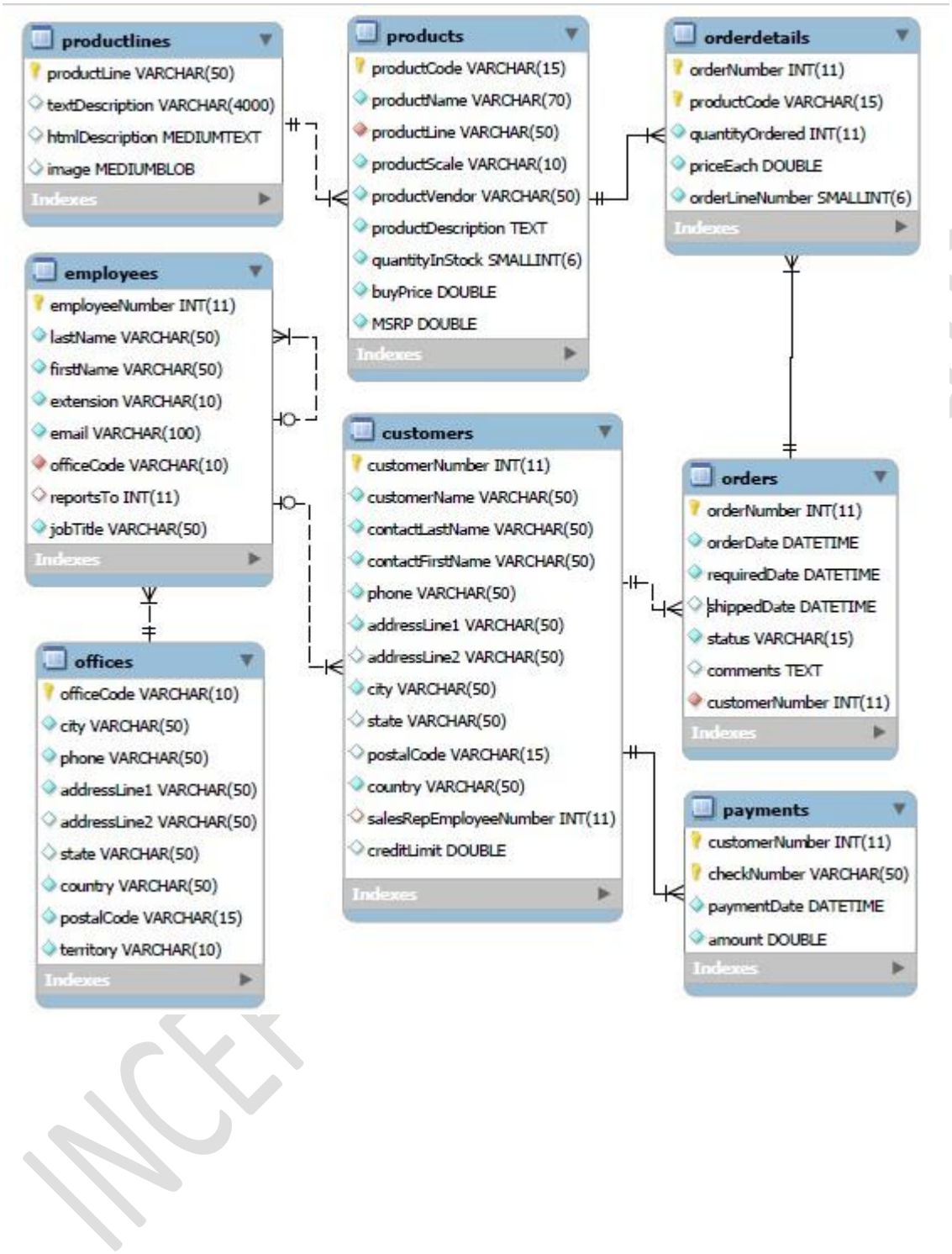
mysql>

source /home/hduser/retailorders/ordersproduct_ORIG.sql

source /home/hduser/retailorders/custpayments_ORIG.sql

source /home/hduser/retailorders/empoffice.sql

Entity Relationship (ER) Diagram of the Tables Schema:



Raw Layer:

SQOOP:

Rather than passing password inline with the sqoop command, 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, direct mode, ~ delimiter, reading without splitby column with primary key as split by

Import the employees table from the DB –

Change the last value according to the data present in the upddt column of employees table in MYSQL, this is to later automate the process.

```
sqoop --options-file /home/hduser/retailorders/empofficeoption --password-file
/user/hduser/retailorders/root.password \
-table employees -m 2 --target-dir /user/hduser/retailorders/employees \
-fields-terminated-by '~' --lines-terminated-by '\n' --incremental lastmodified --check-column upddt --last-value
'2020-10-01' \
--merge-key employeenumber;

sqoop create-hive-table --connect jdbc:mysql://127.0.0.1/empoffice --username root --password root --table
employees \
--hive-table retailstg.employees --fields-terminated-by '~';

load data inpath '/user/hduser/retailorders/employees' overwrite into table retailstg.employees;
```

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 /user/hduser/retailorders/offices/  --fields-terminated-by '~' --lines-terminated-by '\n'
```

```
sqoop create-hive-table --connect jdbc:mysql://127.0.0.1/empoffice --username root --password root \
--table offices --hive-table retailstg.offices --fields-terminated-by '~'
```

```
load data inpath '/user/hduser/retailorders/offices' overwrite into table retailstg.offices;
```

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.postalCod
e,c.country ,c.salesRepEmployeeNumber,c.creditLimit ,p.checknumber,p.paymentdate,p.amount  from
customers c inner join payments p on c.customernumber=p.customernumber and year(p.paymentdate)=2020
and month(p.paymentdate)=10 where $CONDITIONS' \
--split-by c.customernumber --delete-target-dir --target-dir custdetails/2020-10/ --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.produ
ctVendor,p.productDescription,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 and
year(o.orderdate)=2020 and month(o.orderdate)=10 where $CONDITIONS' \
--split-by o.customernumber --delete-target-dir --target-dir orderdetails/2020-10/ --null-string 'NA' \
--direct --num-mappers 4 --fields-terminated-by '~' --lines-terminated-by '\n';
```

HIVE:

Start Hive metastore in a terminal:

```
hive --service metastore
```

Start hive cli in another terminal:

```
hive
```

1. **Load the cust and order details data into hive tables from the above sqoop imported location.**
2. **Convert the custdetails data into complex data types (struct in hive).**

1. **Managed Table use case:** Under **staging** retailstg Database **create temporary managed tables**, Load the sqoop imported data into the below managed table **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 as given below using the truncate statement.**

```
create database retailstg;
use retailstg;

create table custdetails_raw(customerNumber string,customerName string,contactFirstName
string,contactLastName string,phone bigint,addressLine1 string,city string,state string,postalCode bigint,country
string,salesRepEmployeeNumber string,creditLimit float,checknumber string,paymentdate date, checkamt float)
row format delimited fields terminated by '~'
location '/user/hduser/custdetails/2020-10';

truncate table orddetstg;

create table orddetstg (customerNumber string, ordernumber string,orderdate date, shippeddate date,status
string, comments string, quantityordered int,priceeach decimal(10,2),orderlinenumber int, productcode string,
productName STRING,productLine STRING, productScale STRING,productVendor STRING,productDescription
STRING,quantityInStock int,buyPrice decimal(10,2),MSRP decimal(10,2))
row format delimited fields terminated by '~'
location '/user/hduser/orderdetails/2020-10/';
```

(or)

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

$ hive -e "use retailstg; drop table if exists custdetstgcured;"

(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.

```
cd ~/retailorders

vi staging_tbls.hql
create database retailstg;
use retailstg;

create table custdetails_raw (customerNumber string,customerName string,contactFirstName  string,contactLastName string,phone bigint,addressLine1 string,city string,state string,postalCode
bigint,country string,salesRepEmployeeNumber string,creditLimit float,checknumber string,paymentdate date, checkamt float)
row format delimited fields terminated by '~'
```

```
location '/user/hduser/custdetails/2020-10';

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

insert overwrite table custdetstgcured select customernumber,contactfirstname, concat(contactfirstname, ' ', contactlastname) , named_struct('addressLine1', addressLine1, 'city', city, 'state', state,
'postalCode', postalCode, 'country', country, 'phone',phone), creditlimit,checknumber,checkamt,paymentdate from custdetails_raw;

truncate table orddetstg;

create external table orddetstg (customerNumber string, ordernumber string,orderdate date, shippeddate date,status string, comments string, quantityordered int,priceeach decimal(10,2),orderlinenumber
int, productcode string, productName STRING,productLine STRING, productScale STRING,productVendor STRING,productDescription STRING,quantityInStock int,buyPrice decimal(10,2),MSRP decimal(10,2))
row format delimited fields terminated by '~'
location '/user/hduser/orderdetails/2020-10/';
```

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

2. External tables with partition/bucketing usecases : Under retail_curated 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.**

```
dfs -rmr /user/hduser/custorders/custdetpartbuckext

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

create table retail_curated.custdetstgcured(customernumber STRING, customername STRING,
contactfullname string, address
struct<addressLine1:string,city:string,state:string,postalCode:bigint,country:string,phone:bigint>, creditlimit
float,checknum string,checkamt float,paymentdate date)
row format delimited fields terminated by '~'
stored as textfile;
```

Cleanup and Curation

```
insert overwrite table custdetstgcured select coalesce(customernumber,0),contactfirstname,
concat(contactfirstname, ' ', contactlastname) , named_struct('addressLine1', addressLine1, 'city', city, 'state',
state, 'postalCode', postalCode, 'country', country, 'phone',phone), round(creditlimit),
CONCAT(REGEXP_EXTRACT( checknumber ,'[A-Z]+', 0), '-',REGEXP_EXTRACT( checknumber ,'[0-9]+',
0)),round(checkamt),coalesce(paymentdate,current_date) from retailstg.custdetails_raw where
creditlimit>=checkamt;


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,country: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 retailstg.custdetstgcured ;

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

3. Create a final external tables with orc/text format orc 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 database retail_discovery;
use retail_discovery;
create external table custordpartfinal (
```

```
customernumber STRING, customername STRING, contactfullname string,phone bigint,creditlimit float,checknum
string,checkamt float,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.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.paymentdate
from retail_curated.custdetpartbuckext cd inner join retail_curated.orddetpartbuckext o
on cd.customernumber=o.customernumber ;
```

Customer Website viewship pattern and Frustration Scoring use case (Important), use this as one of the usecase for your project when attending interview

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. Dimension data load into Raw layer then to Discovery layer.

```
hadoop fs -mkdir -p /user/hduser/retailorders/dimorders
dfs -put -f /home/hduser/retailorders/orders_rate.csv /user/hduser/retailorders/dimorders/orders_rate.csv
```

2. 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_discovery;
drop table if exists dim_order_rate;

create external table retail_discovery.dim_order_rate (rid int,orddesc varchar(200),comp_cust
varchar(10),siverity int, intent varchar(100))
row format delimited fields terminated by ','
location '/user/hduser/retailorders/dimorders/';
```

3. Create the orddetstg table in staging and load the cust_visits.csv 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.

```
Use retailstg;
drop table if exists cust_navigation;
create table  retailstg.cust_navigation (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/cust_visits.csv' overwrite into table retailstg.
cust_navigation;

//hadoop fs -rmr /user/hduser/custmart/

//drop table if exists retail_curated.cust_navigation;
```

```
create external table if not exists retail_discovery.cust_navigation (customernumber string,navigation_index int,navigation_pg string)
row format delimited fields terminated by ','
location '/user/hduser/custnavigation/';
```

4. 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 into table retail_discovery.cust_navigation
select customernumber,pgnavigationidx,pgnavigation
from retailstg.cust_navigation
lateral view posexplode(pagenavigation) exploded_data1 as pgnavigationidx,pgnavigation;
```

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

What is the First and last page visited and the count of visits.

```
select c1.navigation_pg,count(distinct c1.customernumber) custcnt,'last pagevisited' pagevisit
from retail_discovery.cust_navigation  c1 inner join (select a.customernumber,max(a.navigation_index) as maxnavigation from retail_discovery.cust_navigation a group by a.customernumber) as c2 on (c1.customernumber=c2.customernumber and c1.navigation_index=c2.maxnavigation) group by c1.navigation_pg
union all
select navigation_pg,count(distinct customernumber) custcnt,'first pagevisited' pagevisit
from retail_discovery.cust_navigation
where navigation_index=0
group by navigation_pg;
```

What is the most visited page

```
select navigation_pg,count(customernumber) as cnt
from retail_discovery.cust_navigation
group by navigation_pg
order by cnt desc
limit 1;
```

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

```
create external table  retail_discovery.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_discovery.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 retailstg.cust_navigation o left outer join
retail_discovery.dim_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 |
```

What is the mostly used words by the customer, hence the business can think about resolving those service queries by appointing more man or other resources to reduce those recurring queries to provide better customer experience.

```
create table retailstg.stopwords (words string);

load data local inpath '/home/hduser/retailorders/stopwords' overwrite into table retailstg.stopwords;

select splitwords,count(*) as cnt from
(select customernumber, word as splitwords from retailstg.cust_navigation LATERAL VIEW
explode(split(comments,' ')) w as word ) as q1
where upper(splitwords) not in (select upper(words) from retailstg.stopwords)
group by splitwords
order by cnt desc;
```

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 – Phoenix Integeration:

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

create table retail_discovery.customer_intent(custno string, comments string, orddesc string,comp_cust
varchar(100),severity int,intent string)
STORED BY 'org.apache.phoenix.hive.PhoenixStorageHandler'
TBLPROPERTIES (
 "phoenix.table.name" = "customer_intent",
 "phoenix.zookeeper.quorum" = "localhost",
 "phoenix.zookeeper.znode.parent" = "/hbase",
 "phoenix.zookeeper.client.port" = "2181",
```

```
 "phoenix.rowkeys" = "custno",
 "phoenix.column.mapping" = "rowkey:custno",
"phoenix.table.options" = "SALT_BUCKETS=3");
```

```
insert into table retail_discovery.customer_intent select
o.customernumber,o.comments,r.orddesc,r.comp_cust,r.siverity,r.intent
from retailstg.cust_navigation o left outer join retail_discovery.dim_order_rate r
where o.comments like concat('%',r.orddesc,'%');
```

```
create table retail_discovery.employees (empno int, fullname string, reportsto string,jobtitle string)
STORED BY 'org.apache.phoenix.hive.PhoenixStorageHandler'
TBLPROPERTIES (
 "phoenix.table.name" = "employees",
 "phoenix.zookeeper.quorum" = "localhost",
 "phoenix.zookeeper.znode.parent" = "/hbase",
 "phoenix.zookeeper.client.port" = "2181",
 "phoenix.rowkeys" = "empno",
 "phoenix.column.mapping" = "rowkey:empno");
```

```
insert into table retail_discovery.employees
select employeenumber,concat(firstname, ' ',lastname),reportsto,jobtitle from retailstg.employees;
```

```
create table retail_discovery.orderdetails(cust_ordernumber STRING, shippeddate
date,status string, comments string, quantityordered int,priceeach decimal(10,2),quantityInStock
int,buyPrice decimal(10,2),MSRP decimal(10,2), orderdate date,salesrepemployeenumber int)
STORED BY 'org.apache.phoenix.hive.PhoenixStorageHandler'
TBLPROPERTIES (
 "phoenix.table.name" = "orderdetails",
 "phoenix.zookeeper.quorum" = "localhost",
 "phoenix.zookeeper.znode.parent" = "/hbase",
 "phoenix.zookeeper.client.port" = "2181",
 "phoenix.rowkeys" = "cust_ordernumber",
 "phoenix.column.mapping" = "rowkey: cust_ordernumber ");
```

```
insert into table retail_discovery.orderdetails select concat(o.customernumber, ' -', o.ordernumber),
o.shippeddate, o.status, o.comments, o.quantityordered , o.priceeach, o.quantityInStock, o.buyPrice, o.MSRP,
o.orderdate, c.salesrepemployeenumber
from retailstg.orddetstg o left outer join custdetails_raw c on o.customernumber=c.customernumber
where orderdate is not null;
```

Queries for Analysis:

Calculate the top 3 employees in sales department.

```
select EMPNO,FULLNAME,sum(priceeach) sum_sales
FROM ORDERDETAILS o left join EMPLOYEES e on(o.salesrepemployeenumber=e.empno)
```

```
where e.jobtitle='Sales Rep'
group by empno,fullname
order by sum_sales desc
limit 3;
```

Calculate the status wise sum, max, min, average and number of customers

```
select status,sum(priceeach) sum_sales,max(priceeach) max_price,min(priceeach) min_price,avg(priceeach)
avg_price,count(substr(CUST_ORDERNUMBER,1,instr(CUST_ORDERNUMBER,'-')-1)) total_cust
from ORDERDETAILS
group by status;
```

Calculate the Orders shipped after 5 days from the order taken place.

```
select * from ORDERDETAILS
where SHIPPEDDATE-ORDERDATE>5
and status<>'Disputed';
```

Thank You!!