**Building EDW on Cloudera Hadoop**

The Cloudera Hadoop already has a built-in ER system of orders.

<https://www.cloudera.com/developers/get-started-with-hadoop-tutorial/exercise-1.html>

**Step 1:**

Create databases retail\_stage and retail\_ods and retail\_edw in Hive Environment

Retail\_stage: Will collect all the data from different systems

Retail\_ods: Operation Data store

Retail\_edw: Actual Dimensional Model

Create database using the hive command line.

**Step 2:**

Once the databases are created, the tables should be created.

The tables are created from **retail\_ods** database

**Script:**

CREATE TABLE categories (

category\_id int,

category\_department\_id int,

category\_name string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

CREATE TABLE customers (

customer\_id int,

customer\_fname string,

customer\_lname string,

customer\_email string,

customer\_password string,

customer\_street string,

customer\_city string,

customer\_state string,

customer\_zipcode string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

CREATE TABLE departments (

department\_id int,

department\_name string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

CREATE TABLE orders (

order\_id int,

order\_date string,

order\_customer\_id int,

order\_status string

)

PARTITIONED BY (order\_month string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

CREATE TABLE order\_items (

order\_item\_id int,

order\_item\_order\_id int,

order\_item\_order\_date string,

order\_item\_product\_id int,

order\_item\_quantity smallint,

order\_item\_subtotal float,

order\_item\_product\_price float

)

PARTITIONED BY (order\_month string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

CREATE TABLE orders\_bucket (

order\_id int,

order\_date string,

order\_customer\_id int,

order\_status string

)

CLUSTERED BY (order\_id) INTO 16 BUCKETS

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

CREATE TABLE order\_items\_bucket (

order\_item\_id int,

order\_item\_order\_id int,

order\_item\_order\_date string,

order\_item\_product\_id int,

order\_item\_quantity smallint,

order\_item\_subtotal float,

order\_item\_product\_price float

)

CLUSTERED BY (order\_item\_order\_id) INTO 16 BUCKETS

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

CREATE TABLE products (

product\_id int,

product\_category\_id int,

product\_name string,

product\_description string,

product\_price float,

product\_image string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

**Step 3: Create the EDW table:**

**Script:**

-- Create edw tables (following dimension model)

use retail\_edw;

CREATE TABLE products\_dimension (

product\_id int,

product\_name string,

product\_description string,

product\_price float,

product\_category\_name string,

product\_department\_name string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

CREATE TABLE order\_fact (

order\_item\_order\_id int,

order\_item\_order\_date string,

order\_item\_product\_id int,

order\_item\_quantity smallint,

order\_item\_subtotal float,

order\_item\_product\_price float

)

PARTITIONED BY (product\_category\_department string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

We now have the Fact and dimension tables in the Data Warehouse.

**Step 4:**

**Stage** databases are used to get the data from different systems and then combine it together.

Here we are going to get the data from our mysql system using **sqoop**

**Step 4.1**

Using the SQOOP import-all-table command we will be getting all the tables from mysql DB to HIVE.

Read this: <https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html>

**Script: (Run it in Terminal)**

**sqoop import-all-tables \**

**-m 12 \ -- This is the number of mappers**

**--connect "jdbc:mysql://quickstart.cloudera:3306/retail\_db" \**

**--username=retail\_dba \**

**--password=cloudera \**

**--as-avrodatafile \**

**--warehouse-dir=/user/hive/warehouse/retail\_stage.db \**

**--outdir=/Java**

This script will create the AVRO formatted files in the warehouse directory: **/user/hive/warehouse/retail\_stage.db**

Also in the Cloudera home/java it will create the .avsc files that will have the schema for the tables.

**.avsc Files:** Will have the schema that we need to supply to the external table to read the data.

This takes a long time to execute and is resource intensive.

Here we have mentioned the mappers as 12, so the mapreduce will take the min and max id from the table and then split the data into 12 parts. If nothing is mentioned about the mappers then by default it uses 4.

**Note: Till the mappers finish their job, the reducer won’t start.**

The data now in the retail\_stage will have 12 parts from 0 to 11.

After the sqoop import is complete, then do the following steps:

**Step 4.1.1:**

Now Copy all avsc files using

**Create a directory in the user/cloudera/retail\_stage**

**hadoop fs -put //home/cloudera/\*.avsc /user/cloudera/retail\_stage**

Creating External Tables: Below is the script

CREATE EXTERNAL TABLE categories

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail\_stage.db/categories'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail\_stage/categories.avsc');

CREATE EXTERNAL TABLE customers

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail\_stage.db/customers'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail\_stage/customers.avsc');

CREATE EXTERNAL TABLE departments

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail\_stage.db/departments'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail\_stage/departments.avsc');

CREATE EXTERNAL TABLE orders

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail\_stage.db/orders'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail\_stage/orders.avsc');

CREATE EXTERNAL TABLE order\_items

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail\_stage.db/order\_items'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail\_stage/order\_items.avsc');

CREATE EXTERNAL TABLE products

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail\_stage.db/products'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail\_stage/products.avsc');

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail\_stage.db/orders\_part\_avro'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail\_stage/orders\_part\_avro.avsc');

Check and verify the counts. At this stage, all the counts should match.

There are a lot of good sqoop commands and there are attached to the text file in here

Till Now what has happened:

**Mysql → Sqoop Import → Retail\_stage.db → External Tables**.

**Note:** If you run the sqoop multiple times to the same directory, it will fail saying that the directory already exists.

Now add partition to table Orders\_avro\_part. This is done using the alter command and then insert the data using the below script:

CREATE TABLE orders\_part\_avro (

order\_id int,

order\_date bigint,

order\_customer\_id int,

order\_status string

)

PARTITIONED BY (order\_month string)

STORED AS AVRO

LOCATION 'hdfs:///user/hive/warehouse/retail\_stage.db/orders\_part\_avro'

TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart.cloudera/user/cloudera/retail\_stage/orders\_part\_avro.avsc');

-- Adding partition manually

alter table orders\_part\_avro add partition (order\_month='2014-01');

**-- Inserting data to a partition**

insert into table orders\_part\_avro partition (order\_month='2014-01')

select \* from orders where from\_unixtime(cast(substr(order\_date, 1, 10) as int)) like '2014-01%';

drop table orders\_part\_avro

--Inserting into table dynamically

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

insert into table orders\_part\_avro partition (order\_month)

select order\_id, order\_date, order\_customer\_id, order\_status,

substr(from\_unixtime(cast(substr(order\_date, 1, 10) as int)), 1, 7) order\_month from orders;

**Step 5: SQOOP EXPORT**

There could be a multitude of reasons why we need to export data to a traditional system.

The below script will export data from the Hive.

1. Create a reporting database in mysql
2. Use the sqoop command to export data ( Sqoop does not understand HIVE tables. It only understands HDFC directory.

**Script:**

--Connect to mysql and create database for reporting database

--user:root, password:hadoop

mysql -u root -p

create database retail\_rpt\_db;

grant all on retail\_rpt\_db.\* to retail\_dba;

flush privileges;

use retail\_rpt\_db;

create table departments as select \* from retail\_db.departments where 1=2;

Exit;

**SQOOP EXPORT**

--For certification change database name retail\_rpt\_db to retail\_db

sqoop export --connect "jdbc:mysql://quickstart.cloudera:3306/retailrpt" \

--username retail\_dba \

--password cloudera \

--table departments \

--export-dir /apps/hive/warehouse/retail\_ods.db/departments \

--input-fields-terminated-by '|' \

--input-lines-terminated-by '\n' \

--num-mappers 2 \

--batch \

--outdir java\_files

**STEP 6: PUSH STRATEGY**

Mission critical systems will not allow the Sqoop to connect to their systems and fetch the data. Instead, we can use mysql to create the output files from the table and then use those files to load the tables.

**IMP: Hive load does not use mapreduce and hence it cannot validate data for partitioning.**

**Script:**

**#Make sure you understand table structure, delimiter, partition etc, run mysql export command**

**# You need to give the rights for writing the table to the retail\_dba, or else there will be an error saying that you cannot write to that location.**

**The script to give permission is:**

**update mysql.user set file\_priv = 'Y' where user = 'retail\_dba'**

**Commit and restart the mysqlserver**

**>** sudo service mysqld restart

**/\*SQL EXPORT SCRIPTS\*/**

select \* from categories into outfile '/tmpl/categories01.psv' fields terminated by '|' lines terminated by '\n';

select \* from customers into outfile '/tmp/customers.psv' fields terminated by '|' lines terminated by '\n';

select \* from departments into outfile '/tmp/departments.psv' fields terminated by '|' lines terminated by '\n';

select \* from products into outfile '/tmp/products.psv' fields terminated by '|' lines terminated by '\n';

The table creation should be out of the Cloudera file system

**#We cannot use orders and order\_items directly as tables in hive database retail\_ods are partitioned**

**We are loading 4 tables - Categories , Customers , department and products.**

**The orders table is partitioned and hence cannot be loaded in this way**

**Once the temp files are created move them to the HDFS ecosystem using the below commands**

hadoop fs -mkdir /user/root/departments

hadoop fs -put /tmp/departments.psv /user/root/departments

hadoop fs -ls /user/root/department

**Upload the files using Hue**

After the files are uploaded, then start loading the data using the below command

load data inpath '/user/root/departments/\*' overwrite into table departments;

load data inpath '/user/root/customers/\*' overwrite into table customers;

load data inpath '/user/root/categories/\*' overwrite into table categories;

load data inpath '/user/root/products/\*' overwrite into table products;

**Step 7: LOADING THE ORDERS TABLE WHICH IS PARTITIONED**

**#Create orders\_stage under hive database retail\_stage**

hive

use retail\_stage;

CREATE TABLE orders\_stage (

order\_id int,

order\_date string,

order\_customer\_id int,

order\_status string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE;

**Load the data:**

load data local inpath '/tmp/orders.psv' overwrite into table orders\_stage;

**7.1** Inserting data into retail\_ods.orders which is partitioned by month.

The dynamic partition must be enabled, as it is false by default

Below is the script:

set hive.exec.dynamic.partition.mode=nonstrict

insert overwrite table retail\_ods.orders partition (order\_month)

select order\_id, order\_date, order\_customer\_id, order\_status,

substr(order\_date, 1, 7) order\_month from retail\_stage.orders\_stage;

This query will load the data automatically into the partitions as it uses map reduce

**#Now we have 2 tables retail\_stage.order\_items and retail\_stage.orders**

**#We need to join these 2 and populate retail\_ods.order\_items table which have additional columns**

**#order\_item\_order\_date and order\_month**

**#Also table is partitioned by order\_month**

insert overwrite table order\_items partition (order\_month)

select oi.order\_item\_id, oi.order\_item\_order\_id, o.order\_date,

oi.order\_item\_product\_id, oi.order\_item\_quantity, oi.order\_item\_subtotal,

oi.order\_item\_product\_price, substr(o.order\_date, 1, 7)

order\_month from retail\_stage.order\_items oi join retail\_stage.orders\_stage o

on oi.order\_item\_order\_id = o.order\_id;

Now we insert data into the bucketed tables.

The bucketed tables are used to improve performance of the HIVEQL

**Script:**

set hive.enforce.bucketing =true;

insert overwrite table orders\_bucket

select o.order\_id, o.order\_date, o.order\_customer\_id,o.order\_status

from orders o;

This Completed all the retail\_stage and retail\_ods data population. After this we need to load the data into the Fact and Dimensions Tables.

**Step 8: Final Loading of data into Facts and Dimensional tables**

**Script to load the Dimension Table:**

insert overwrite table retail\_edw.products\_dimension

select p.product\_id,

p.product\_name,

p.product\_description,

p.product\_price,

c.category\_name as product\_category\_name,

d.department\_name as product\_department\_name

from retail\_ods.products p join retail\_ods.categories c on p.product\_category\_id = c.category\_id

join retail\_ods.departments d on c.category\_department\_id = d.department\_id

Same way the Facts table can be loaded.