**Big Data Management Hive Lab**

In Session 5 of “Big Data Management”, we will experiment with **Hive** for executing SQL queries on Hadoop MapReduce. The tutorial draws on examples in Chapter 17 of *Hadoop: The Definitive Guide*. The dataset is a dimensional model (a “star schema”) for a sample Orders dataset, with Day, Product, and Customer dimensions. Log in to the Cloudera VM and get the data with:

git clone https://github.com/SubbaReddyYeruva/abc

Take a look around the files with “head”. You can also check their length with “wc -l”:

cd orders\_mart

head ORDER\_FACT.csv

wc -l ORDER\_FACT.csv

…

You can see that these are comma-separated values (CSV) files and that they have header rows.

Move all the files into the Hadoop filesystem, HDFS:

hdfs dfs -mkdir orders

hdfs dfs -put ORDER\_FACT.csv orders/

hdfs dfs -put PRODUCT\_DIMENSION.csv orders/

hdfs dfs -put DAY\_DIMENSION.csv orders/

hdfs dfs -put CUSTOMER\_DIMENSION.csv orders/

hdfs dfs -ls orders/

**Loading Tables into Hive**

In Hive, you first define a table’s schema with something that looks like a SQL “CREATE TABLE” statement, before loading data. No data is transformed yet.

Type “hive” to enter the Hive console.

Now type the following just to look around:

show databases;

show tables;

Create the first table:

CREATE TABLE customers (ckey INT, name STRING, sex STRING, age INT, state STRING, age\_range STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

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

No data has moved yet. You can use “describe” to find out about the schema of the table.

DESCRIBE customers;

To actually load the data, you’ll do the following:

LOAD DATA INPATH '/user/cloudera/orders/CUSTOMER\_DIMENSION.csv' OVERWRITE INTO TABLE customers;

This is a “Hive managed” table. What that means is that if you DROP the table, the data will be deleted forever. The other option is an external table, which lives on outside of Hadoop. If you DROP an external table, only the metadata in the Hive metastore is lost.

Now notice what happened to the file in HDFS:

hdfs dfs -ls orders

The CUSTOMER\_DIMENSION.csv file was moved. To where?

hdfs dfs -ls /user/hive/warehouse

Aha! When Hive “manages” a file, it moves it into its own directory.

Let’s also get the product and fact table.

CREATE TABLE products (pkey INT, product STRING, brand STRING, category STRING, price FLOAT, cost FLOAT)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

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

LOAD DATA INPATH '/user/cloudera/orders/PRODUCT\_DIMENSION.csv' OVERWRITE INTO TABLE products;

CREATE TABLE orders (dkey INT, ckey INT, orderid INT, pkey INT, qty INT, dollars DECIMAL(10,2), cost DECIMAL(10,2))

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

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

LOAD DATA INPATH '/user/cloudera/orders/ORDER\_FACT.csv' OVERWRITE INTO TABLE products;

**Column-oriented data formats and CTAS:**

We don’t have to store our data in plain-text files. Hive can use several row-oriented or column-oriented data formats. One of the column-oriented formats is Parquet, and column-orientation is useful for typical analytical queries that focus on one or two columns across many rows at once.

This conversion illustrates the common “CTAS” construct: CREATE TABLE … AS… SELECT …

CREATE TABLE orders\_parquet STORED AS PARQUET

AS

SELECT \* FROM orders;

**Multi-table insert**

Unlike your typical database, we can produce two outputs in one query. Just like with Pig, this minimizes the use of unnecessary MapReduce steps. We can read the data once to write it twice. Create two empty tables with the same schema as “customers”:

FROM customers

INSERT OVERWRITE TABLE young\_customers

SELECT \* WHERE age<21

INSERT OVERWRITE TABLE old\_customers

SELECT \* WHERE age>50;

**Some SQL-on-Hadoop queries:**

Now let’s try some queries and see how they’re transformed into MapReduce jobs.

SELECT sex, count(ckey) FROM young\_customers GROUP BY sex;

We can get some information on a query by prefacing a query with the EXPLAIN keyword.

EXPLAIN

SELECT sex, count(ckey) FROM young\_customers GROUP BY sex;

Now let’s do some typical analytical queries… how about finding our top ten customers by revenue?

SELECT customers.ckey, name, sum(dollars)

FROM customers JOIN orders ON customers.ckey=orders.ckey

GROUP BY customers.ckey, name

ORDER BY 3 DESC

LIMIT 10;

How about finding the quantities of each product sold in each state?

SELECT state, product, sum(qty) AS units\_sold

FROM customers

JOIN orders ON customers.ckey = orders.ckey

JOIN products ON orders.pkey = products.pkey

GROUP BY state, product

Or we can find the best selling product brands in Arizona:

SELECT brand, sum(dollars)

FROM orders

JOIN customers ON orders.ckey = arizonans.ckey

JOIN products ON orders.pkey=products.pkey

WHERE customers.state==AZ

GROUP BY brand

ORDER by 2 DESC;