Table of Contents

Hive training material

Hive

Introduction to hive

Slides.

Installation

Follow the following steps to install hive on your machine

Step 1: Download tar.gz from apache site

Download from https://hive.apache.org/downloads.html

Step 2: Extract tar.gz

```
tar -zxvf apache-hive-0.14.0-bin.tar.gz
```

Step 3 : Start hive shell

```
bin/hive
```

HQL

HQL handson.

DDL

The following commands are Hive DDL

Create table

```
create table sales(transactionId string,
customerId String, itemId string, amount double)
row format delimited fields terminated by ","
stored as textfile;
```

Show tables

```
show tables;
```

Describe

```
describe sales;
```

Drop table

```
drop sales;
```

Behind the scenes

Observe a folder creater in */user/hive/warehourse/sales* in hdfs.

Loading data

Load from local file system

```
load data local inpath '/home/hadoop/sales.csv'
overwrite into table sales;
```

Load from hdfs

```
load data inpath '/user/hadoop/sales.csv'
overwrite into table sales;
```

Observe the folder structure in hdfs

External table

External table is creating a table for the data already present in hdfs, to demonstrate this lets put the files to HDFS first

```
bin/hdfs dfs -put /home/hadoop/sales.csv /user/hadoop/Sales_External
```

Now we point this directory for the table we are going to create

```
create external table sales_external(transactionId string,
customerId String, itemId string, amout double)
row format delimited fields terminated by ","
stored as textfile location '/user/hadoop/Sales_External';
```

External table only takes folders.

Deleting tables vs external tables

Querying

Select

```
select * from sales;
```

It donot run map/reduce.

Projection

```
select itemId from sales;
```

Where condition

```
select * from sales where customerId='1';
```

Aggregation

```
select count(*) from sales;
```

```
select sum(amount) from sales;
```

Observe map/reduce on application master.

Group by

```
select customerId,count(*) from
sales group by customerId;
```

Joins

```
 create table customer(customerId String,
 customerName String) row format
 delimited fields terminated by ","
 stored as textfile;
```

```
load data local inpath '/home/hadoop/customers.csv'
overwrite into table customer;
```

```sql
select * from sales join customer
on (customer.customerId=sales.customerId);
```

Partitioning

Partitioning allows you to segreate data by specific column. Segregate sales data by the dates.

Create partitioned table

```
create table sales_partitioned(transactionId string,
customerId String, itemId string, amount double)
partitioned by (month string)
row format delimited fields
terminated by "," stored as textfile;
```

Describe

```
desc sales_partitioned;
```

Load data into partitions

Get data from here

```
load data local inpath '/home/hadoop/sales_APR.csv'
into table sales_partitioned partition(month='APR');
```

Observe hdfs directory structure

```
load data local inpath '/home/hadoop/sales_MAY.csv'
into table sales_partitioned partition(month='MAY');
```

Look into the contents of directory **/user/hive/warehouse/sales_partitioned**

Query using partition

```
select count(*) from sales_partitioned where month='MAY';
```

NestedQueries (Sub queries)

Subqueries in the FROM Clause

Syntax

```
SELECT ... FROM (subquery) name ...
SELECT ... FROM (subquery) AS name ...
```

The subquery has to be given a name because every table in a FROM clause must have a name. Columns in the subquery select list must have unique names.

Get all the customers who's total sales is greater than 600;

```
 select customerId,total_amount from
 ( select customerid, sum(amount) total_amount
 from sales group by customerid ) t2
 where total_amount >=600;
```

Subqueries in the WHERE Clause

IN and NOT IN statements

called uncorrelated subqueries because the subquery does not reference columns from the parent query

Check for transactions from customer where there is an entry in the customer row

```
 select * from sales where customerId
 in ( select customerId from customer);
```

EXISTS and NOT EXISTS statements

called correlated subqueries because the subquery references columns from the parent query

List customer who has made atleast one transaction

```
select * from customer where exists
( select transactionId from sales where customer.customerId = sales.customerId);
```