Experiment 5: HIVE

Apache Hive Introduction

Apache Hive is an enterprise data warehouse system used to query, manage, and analyse data stored in the Hadoop Distributed File System

The Hive Query Language (HiveQL) facilitates queries in a Hive command-line interface shell. Hadoop can use HiveQL as a bridge to communicate with relational database management systems and perform tasks based on SQL-like commands.

Install Apache Hive on Ubuntu 20.04.

Prerequisites

Apache Hive is based on Hadoop and requires a fully functional Hadoop framework.

5.1 Install Apache Hive on Ubuntu

To configure Apache Hive, first you need to download and unzip Hive. Then you need to customize the following files and settings:

- Edit **.bashrc** file
- Edit **hive-config.sh** file
- Create **Hive directories** in HDFS
- Configure **hive-site.xml** file
- Initiate **Derby database**

Step 1: Download and Untar Hive

Access your Ubuntu command line and download the compressed Hive files using and the wget command followed by the download path:

$ wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz

```
hdoop@phoenixnap:~$ wget https://downloads.apache.org/hive/hive-3.1.2/apache-hi
ve-3.1.2-bin.tar.gz
--2020-06-01 08:11:30--  https://downloads.apache.org/hive/hive-3.1.2/apache-hi
ve-3.1.2-bin.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 88.99.95.219, 2a01:4f8
:10a:201a::2
Connecting to downloads.apache.org (downloads.apache.org)|88.99.95.219|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 278813748 (266M) [application/x-gzip]
Saving to: 'apache-hive-3.1.2-bin.tar.gz'

apache-hive-3.1.2-b 100%[===================>] 265.90M  10.9MB/s    in 25s

2020-06-01 08:11:55 (10.7 MB/s) - 'apache-hive-3.1.2-bin.tar.gz' saved [2788137
48/278813748]
```

Once the download process is complete, untar the compressed Hive package:

Step 2: Configure Hive Environment Variables (bashrc)

The **$HIVE_HOME** environment variable needs to direct the client shell to the *apache-hive-3.1.2-bin* directory. Edit the *.bashrc* shell configuration file using a text editor of your choice (we will be using nano):

$ sudo nano .bashrc

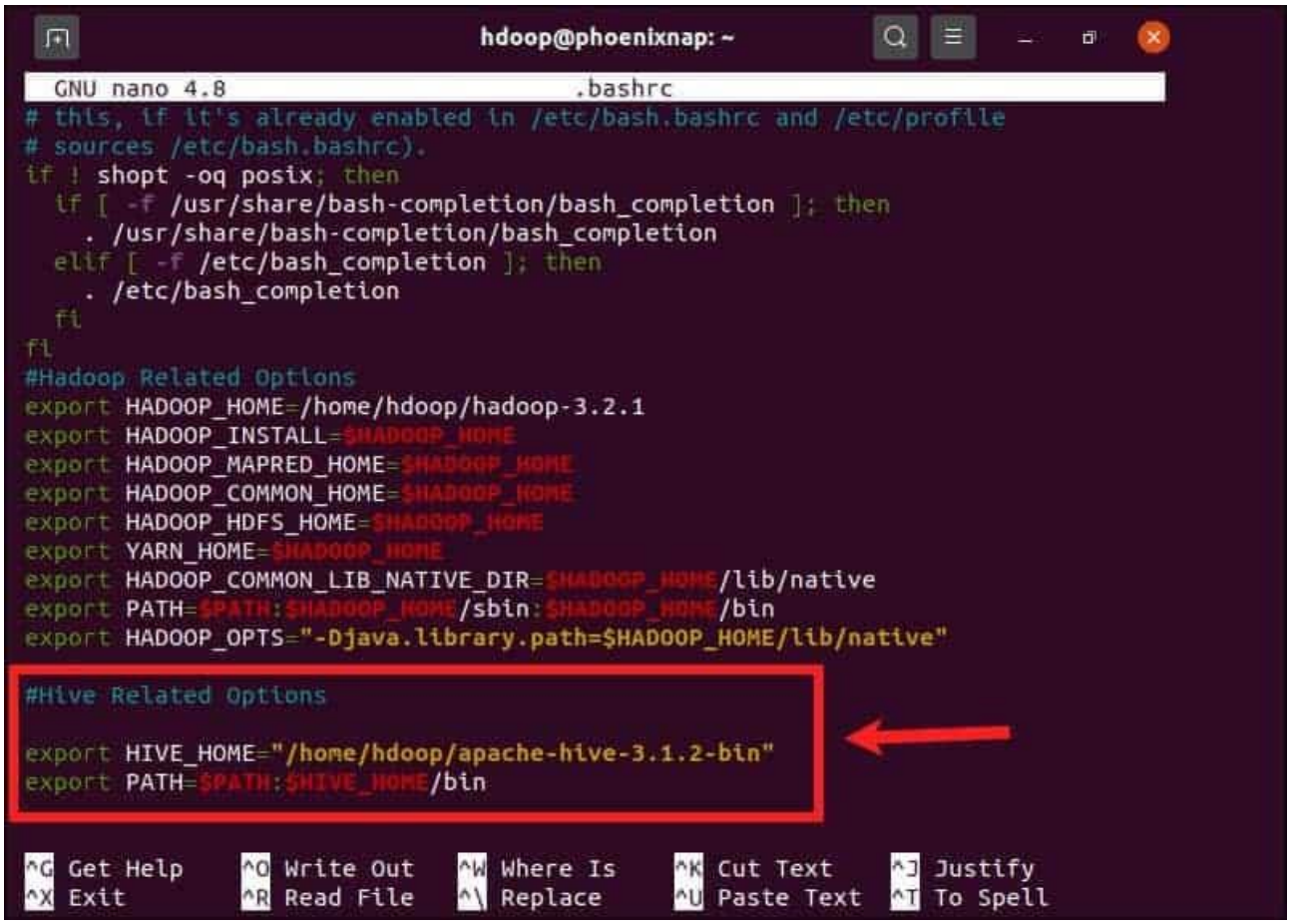
Append the following Hive environment variables to the *.bashrc* file:

export HIVE_HOME= "home/hdoop/apache-hive-3.1.2-bin"

export PATH=$PATH:$HIVE_HOME/bin

The Hadoop environment variables are located within the same file.

$ tar xzf apache-hive-3.1.2-bin.tar.gz



The Hive binary files are now located in the *apache-hive-3.1.2-bin* directory.

Save and exit the *.bashrc* file once you add the Hive variables. Apply the changes to the current environment with the following command:

$ source ~/.bashrc

Step 3: Edit hive-config.sh file

Apache Hive needs to be able to interact with the Hadoop Distributed File System. Access the *hive-config.sh* file using the previously created **$HIVE_HOME** variable:

$ sudo nano $HIVE_HOME/bin/hive-config.sh

Add the **HADOOP_HOME** variable and the full path to your Hadoop directory:

export HADOOP_HOME=/home/hdoop/hadoop-3.2.1

```
# Allow alternate conf dir location.
HIVE_CONF_DIR="${HIVE_CONF_DIR:-$HIVE_HOME/conf}"

export HIVE_CONF_DIR="/home/hdoop/apache-hive-3.1.2-bin/conf"
export HADOOP_HOME=/home/hdoop/hadoop-3.2.1
```

Save the edits and exit the *hive-config.sh* file.

Step 4: Create Hive Directories in HDFS

Create two separate directories to store data in the HDFS layer:

- The temporary, *tmp* directory is going to store the intermediate results of Hive processes.
- The *warehouse* directory is going to store the Hive related tables.

Create tmp Directory

Create a *tmp* directory within the HDFS storage layer. This directory is going to store the intermediary data Hive sends to the HDFS:

$ hdfs dfs -mkdir /tmp

Add write and execute permissions to tmp group members:

$ hdfs dfs -chmod g+w /tmp

Check if the permissions were added correctly:

$ hdfs dfs -ls /

The output confirms that users now have write and execute permissions.

```
hdoop@phoenixnap:~$ hdfs dfs -ls /
Found 4 items
drwxr-xr-x   - hdoop supergroup          0 2020-06-02 02:37 /Exampledir
drwxrwxr-x   - hdoop supergroup          0 2020-06-02 07:26 /tmp
```

Create warehouse Directory

Create the *warehouse* directory within the */user/hive/* parent directory:

$ hdfs dfs -mkdir -p /user/hive/warehouse

Add **write** and **execute** permissions to *warehouse* group members:

$ hdfs dfs -chmod g+w /user/hive/warehouse

Check if the permissions were added correctly:

$ hdfs dfs -ls /user/hive

The output confirms that users now have write and execute permissions.

```
hdoop@phoenixnap:~$ hdfs dfs -ls /user/hive
Found 1 items
drwxrwxr-x   - hdoop supergroup          0 2020-06-02 09:06 /user/hive/warehous
e
```

Step 5: Configure hive-site.xml File (Optional)

Apache Hive distributions contain template configuration files by default. The template files are located within the Hive *conf* directory and outline default Hive settings.

Use the following command to locate the correct file:

$ cd $HIVE_HOME/conf

List the files contained in the folder using the **ls** command.

```
hdoop@phoenixnap:~$ cd $HIVE_HOME/conf
hdoop@phoenixnap:~/apache-hive-3.1.2-bin/conf$ ls
beeline-log4j2.properties.template
hive-default.xml.template          ivysettings.xml
hive-env.sh.template               llap-cli-log4j2.properties.template
hive-exec-log4j2.properties.template  llap-daemon-log4j2.properties.template
hive-log4j2.properties.template    parquet-logging.properties
```

Use the *hive-default.xml.template* to create the *hive-site.xml* file:

$ cp hive-default.xml.template hive-site.xml

Access the *hive-site.xml* file using the nano text editor:

$ sudo nano hive-site.xml

Using Hive in a stand-alone mode rather than in a real-life Apache Hadoop cluster is a safe option for newcomers. You can configure the system to use your local storage rather than the HDFS layer by setting the *hive.metastore.warehouse.dir* parameter value to the location of your Hive *warehouse* directory.

```
GNU nano 4.8                    hive-site.xml                    Modified
  <property>
    <name>hive.metastore.db.type</name>
    <value>DERBY</value>
    <description>
      Expects one of [derby, oracle, mysql, mssql, postgres].
      Type of database used by the metastore. Information schema &amp;amp; JDBCSto>
    </description>
  </property>
  <property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
    <description>location of default database for the warehouse</description>
  </property>
  <property>
    <name>hive.metastore.warehouse.external.dir</name>
    <value/>
    <description>Default location for external tables created in the warehouse>
  </property>
  <property>
    <name>hive.metastore.uris</name>
    <value/>
    <description>Thrift URI for the remote metastore. Used by metastore client>
  </property>
  <property>
    <name>hive.metastore.uri.selection</name>
```

Step 6: Initiate Derby Database

Apache Hive uses the Derby database to store metadata. Initiate the Derby database, from the Hive *bin* directory using the **schematool** command:

$ HIVE_HOME/bin/schematool –initSchema –dbType derby

The process can take a few moments to complete.

```
Initialization script completed
schemaTool completed
hdoop@phoenixnap:~/apache-hive-3.1.2-bin/bin$
```

Derby is the default metadata store for Hive. If you plan to use a different database solution, such as MySQL or PostgreSQL, you can specify a database type in the *hive-site.xml* file.

How to Fix guava Incompatibility Error in Hive

If the Derby database does not successfully initiate, you might receive an error with the following content:

"Exception in thread "main" java.lang.NoSuchMethodError: com.google.common.base.Preconditions.checkArgument(ZLjava/lang/String;Ljava/lang/Object;)V"

This error indicates that there is most likely an incompatibility issue between Hadoop and Hive *guava* versions.

Locate the **guava jar** file in the Hive *lib* directory:

$ ls $HIVE_HOME/lib

Locate the **guava jar** file in the Hadoop *lib* directory as well:

$ ls $HADOOP_HOME/share/hadoop/hdfs/lib

The two listed versions are not compatible and are causing the error. Remove the existing **guava** file from the Hive *lib* directory:

$ rm $HIVE_HOME/lib/guava-19.0.jar

Copy the **guava** file from the Hadoop *lib* directory to the Hive *lib* directory:

$ cp $HADOOP_HOME/share/hadoop/hdfs/lib/guava-27.0-jre.jar $HIVE_HOME/lib/

Use the **schematool** command once again to initiate the Derby database:

$ HIVE_HOME/bin/schematool –initSchema –dbType derby

Launch Hive Client Shell on Ubuntu

Start the Hive command-line interface using the following commands:

$ cd $HIVE_HOME/bin

hive

You are now able to issue SQL-like commands and directly interact with HDFS.

```
hdoop@phoenixnap:~$ cd $HIVE_HOME/bin
hdoop@phoenixnap:~/apache-hive-3.1.2-bin/bin$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hdoop/apache-hive-3.1.2-bin/lib/log4j-s
lf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hdoop/hadoop-3.2.1/share/hadoop/common/
lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation
.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 43f09b9d-bc36-4e29-a1d6-045a92e66b98

Logging initialized using configuration in jar:file:/home/hdoop/apache-hive-3.1
.2-bin/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versi
ons. Consider using a different execution engine (i.e. spark, tez) or using Hiv
e 1.X releases.
Hive Session ID = 39a404e5-f53f-47c4-9ff7-7581a413edd6
hive>
```

Start-all.sh

5.2 HIVE - List of Statements

start-all.sh	* how to start all nodes in hadoop *
jps	* check the status *
hive	* start Hive *
show databases;	* lists all existing database*
create database if not exists HOSPITAL;	* creates a database *
use HOSPITAL;	* get into the database *

show tables; * lists all the tables in a database *

How to create internal /managed tables

create table patient(pid int,pfname string, age int,plname string,state string,reason string) row format delimited fields terminated by '/t' ;

LOAD DATA local INPATH '/home/hadoop/Documents/patient.txt' into table patient;

desc student;

create table app(pid int,did string,dname string,rating int,specialization string,hid string)row format delimited fields terminated by '\t';

How to insert values into a table

LOAD DATA local INPATH '/home/hadoop/Documents/app.txt' into table app;

How to create External tables

create external table emp(eid int,ename string,rating float,department string,lname string,state string)row format delimited fields terminated by ',' stored as textfile;

show tables; ** lists the tables in the database**

desc emp; * gives the structure of the table *

How to insert values into a table

LOAD DATA local INPATH '/home/hadoop/Documents/emp1.txt' into table emp;

OR

insert into emp values('eid','ename','rating','department','lname','state');

How display all values in a table

select * from emp;

How to drop a table

drop table emp;

HOW TO ADD COLUMN TO A EXISTING TABLE

alter table emp add columns (age int);

HOW TO DROP COLUMNS

alter table emp replace columns(sid int,sname string,grade float,department string,lname string,state string);

SAMPLE QUERIES

1. select * from patient;
2. select * from patient where age > 60 and reason<>'fever';
3. select did, dname, (rating + 1.0) AS raise from app;
4. select * from patient where reason=’cold’ or reason='fever';
5. Select dname from app where did in(1,2,3);
6. select max(rating) from app;
7. select min(age) from patient;
8. select max(rating) from app group by specialization; ent 4.8 derma 3.8 gyn 4.5
9. select avg(rating) from app;
10. select sum(rating) from app where specialization=’cardiologist’;

case statement

11. select dname,rating,case when rating<=2 then 'low' when rating >=3 and grade <=4 then 'average' when grade>=5 then 'excellent' else 'not valid' end as rating_range from app; john 1 not valid
12. select substr(dname,2,3) from app;

13. select concat(did,'_',dname) from app;

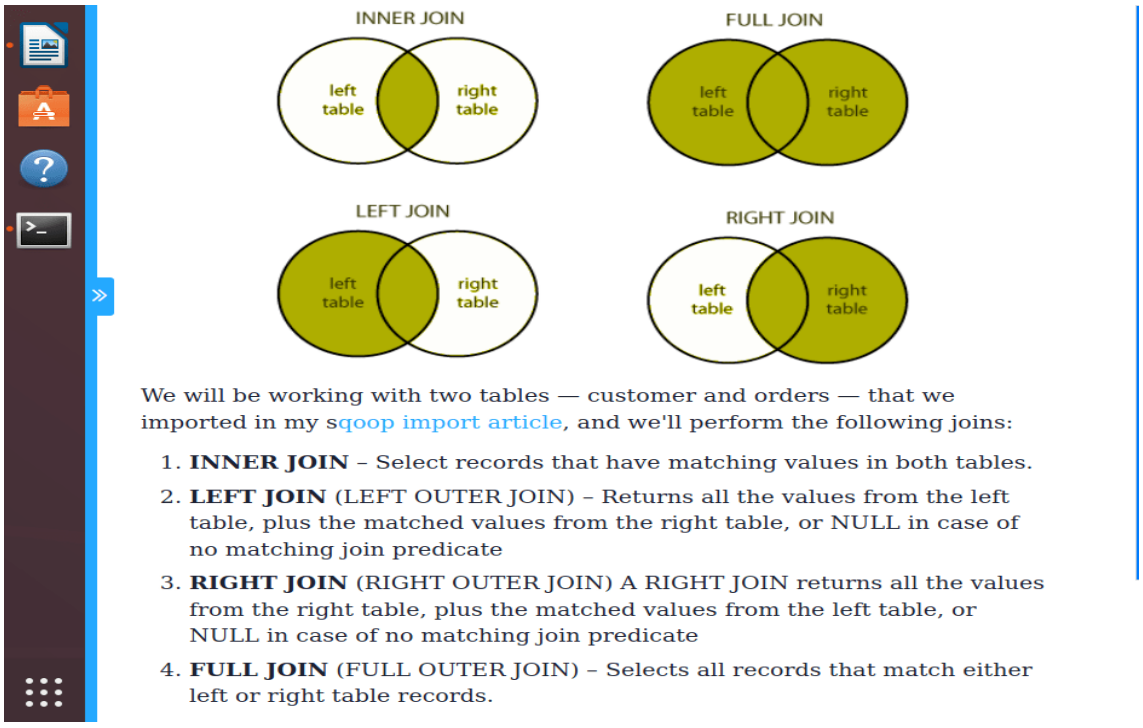
JOIN

14. select p.pid,p.pfname,a.dname from patient p join app a on (p.pid=a.pid);

15. select p.pid,p.pfname , a.dname from patient p left outer join app a on (p.pid=a.pid);

16. select p.pid,p.pfname , a.dname from patient p right outer join app a on (p.pid=a.pid);

17. select p.pid,p.pfname , a.dname from patient p full outer join app a on (p.pid=a.pid);

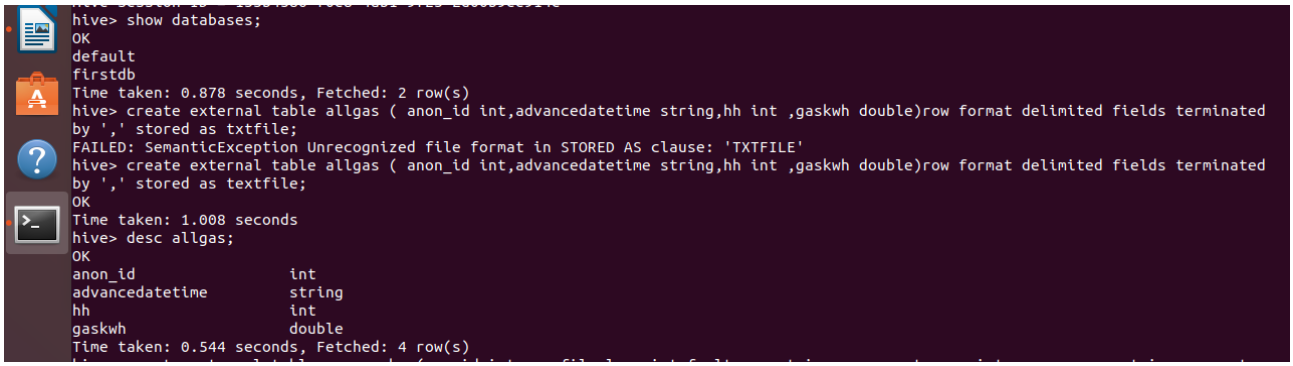


We will be working with two tables — customer and orders — that we imported in my sqoop import article, and we'll perform the following joins:

1. **INNER JOIN** – Select records that have matching values in both tables.
2. **LEFT JOIN** (LEFT OUTER JOIN) – Returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching join predicate
3. **RIGHT JOIN** (RIGHT OUTER JOIN) A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate
4. **FULL JOIN** (FULL OUTER JOIN) – Selects all records that match either left or right table records.

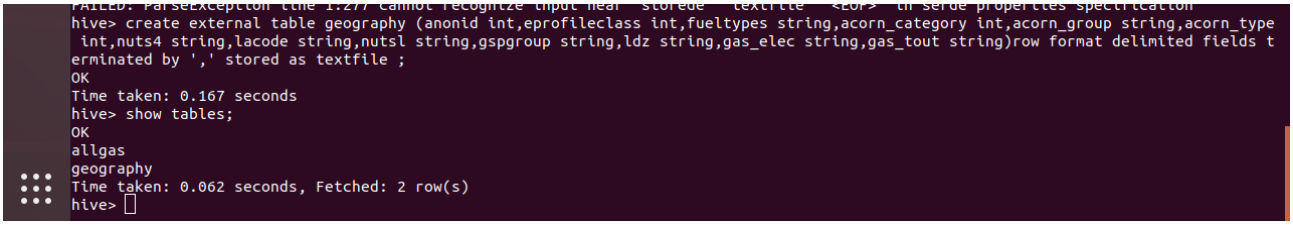
Screenshots

1. Show databases;



2. desc table_name;

3. create external table; (note without the keyword external it will be internal table)



4 . Show tables;

5. Load values into table

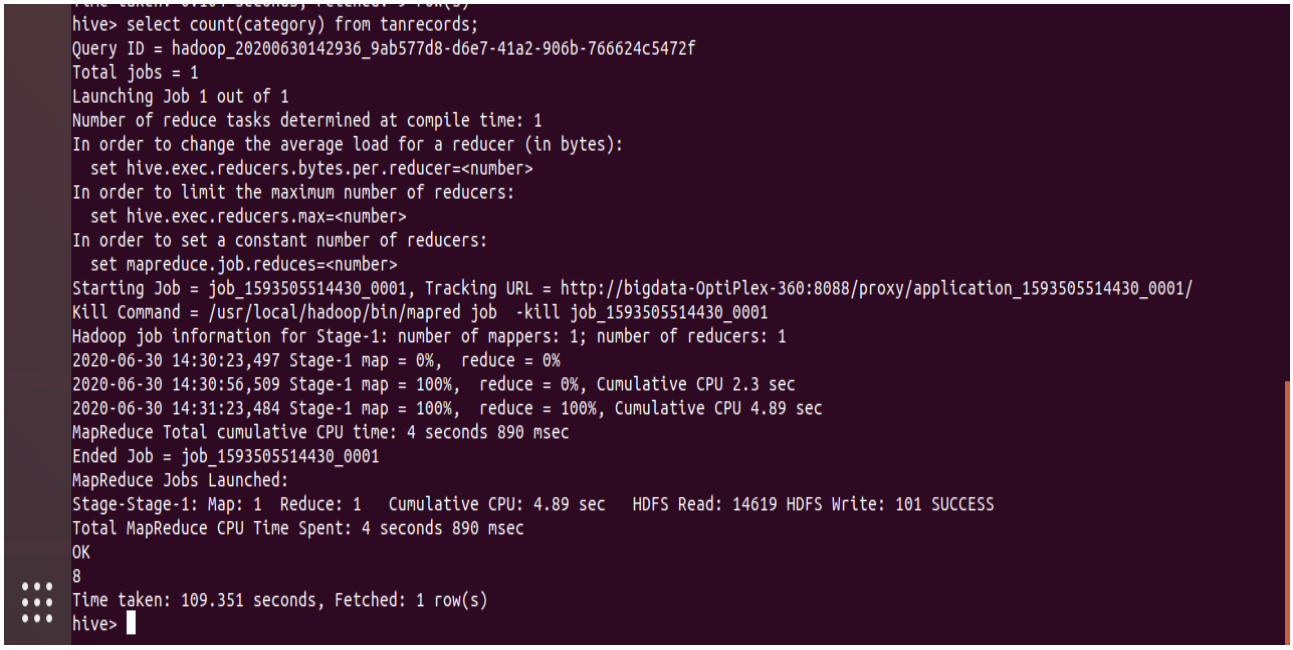
```
hive> create table tanrecords(txno int,txndate string,custno int,amount double,category string,product string,city string,state strin
g,spendby string) row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 0.823 seconds
hive> show tables;
OK
tanrecords
Time taken: 0.035 seconds, Fetched: 1 row(s)
hive> select * from tanrecords;
OK
Time taken: 0.318 seconds
hive> load data local inpath '/home/hadoop/Documents/custtxn.txt' into table tanrecords;
Loading data to table trial.tanrecords
OK
Time taken: 2.039 seconds
hive> select * from tanrecords;
OK
NULL    NULL    NULL    NULL    NULL    NULL    NULL    NULL    NULL
1       12/05/2020      100     1234567.0       electronics     laptop  bangalore       karnataka       manager
2       31/02/2019      101     1234567.0       cloths  top     mangalore       karnataka       manager
3       02/05/2016      102     1234567.0       cloths  pant    bombay  maharastra      manager
4       12/05/1998      103     1234567.0       electronics     watch   mangalore       karnataka       manager
5       19/05/1994      104     1234567.0       electronics     tv      bangalore       karnataka       manager
6       24/04/2005      105     1234567.0       gold    jewlery bombay  maharastra      manager
7       12/06/2004      106     1234567.0       savings education       mysore  karnataka       manager
8       12/06/2015      107     1234567.0       trips   worldtour       bangalore       karnataka       manager
Time taken: 0.197 seconds, Fetched: 9 row(s)
hive>
```

OR

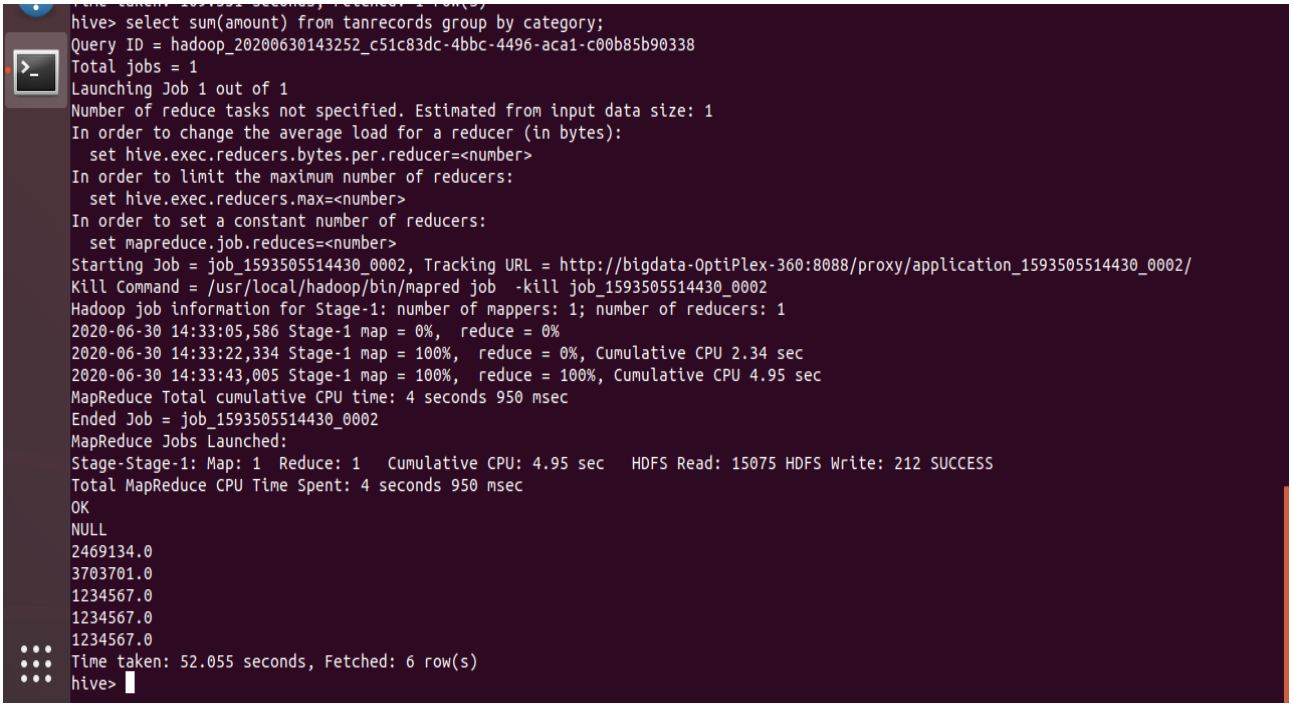
using Insert statement

```
Time taken: 0.25 seconds
hive> LOAD DATA local INPATH '/home/hadoop/Documents/app.txt' into table app ;
Loading data to table hospital.app
OK
Time taken: 0.646 seconds
hive> select * from app;
OK
1       d1      sam     2       Immunologists   H1
2       d2      mary    3       Cardiologists   H1
3       d3      john    4       Dermatologists  H2
4       d4      alex    5       Endocrinologists        H2
5       d5      rose    5       Family Physicians       H3
6       d6      richard 3       Nephrologists   H3
7       d7      melissa 4       Family Physicians       H3
8       d7      usha    5       Family Physicians       H3
9       d3      tom     3       Dermatologists  H3
10      d4      kyane   4       Immunologists   H2
Time taken: 0.353 seconds, Fetched: 10 row(s)
hive> insert into app values('11','d8','george','3','dermatologist','H4);
FAILED: ParseException line 1:67 character '<EOF>' not supported here
hive> insert into app values('11','d8','george','3','dermatologist','H4');
Query ID = hadoop_20200711182803_ca3cc257-955b-4489-857f-6688e5fb13c9
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1594385239703_0005, Tracking URL = http://bigdata-OptiPlex-360:8088/proxy/application_1594385239703_0005/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1594385239703_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-07-11 18:29:40,123 Stage-1 map = 0%,  reduce = 0%
2020-07-11 18:30:40,535 Stage-1 map = 0%,  reduce = 0%
2020-07-11 18:31:11,590 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.09 sec
2020-07-11 18:32:03,644 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.94 sec
MapReduce Total cumulative CPU time: 6 seconds 940 msec
Ended Job = job_1594385239703_0005
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/hospital.db/app/.hive-staging_hive_2020-07-11_18-28-03_416_2225429
802119696188-1/-ext-10000
Loading data to table hospital.app
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.94 sec   HDFS Read: 20207 HDFS Write: 400 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 940 msec
OK
Time taken: 246.452 seconds
hive>
```

6. Query sample for Count aggregate function



7. Query sample for sum aggregate function



8. Queries using and & or

```
hive> select * from tanrecords where custno='102';
OK
3       02/05/2016      102     1234567.0       cloths  pant    bombay  maharastra      manager
Time taken: 0.602 seconds, Fetched: 1 row(s)
hive> select * from tanrecords where custno='102' or custno='103';
OK
3       02/05/2016      102     1234567.0       cloths  pant    bombay  maharastra      manager
4       12/05/1998      103     1234567.0       electronics     watch   mangalore       karnataka       manager
Time taken: 0.214 seconds, Fetched: 2 row(s)
hive> select * from tanrecords where custno='102' and txno='3';
OK
3       02/05/2016      102     1234567.0       cloths  pant    bombay  maharastra      manager
Time taken: 0.253 seconds, Fetched: 1 row(s)
hive>
```

9. usage of limit keyword

```
hive> select * from tanrecords limit 2;
OK
NULL    NULL    NULL    NULL    NULL    NULL    NULL    NULL    NULL
1       12/05/2020      100     1234567.0       electronics     laptop  bangalore       karnataka       manager
Time taken: 0.165 seconds, Fetched: 2 row(s)
hive> select * from tanrecords limit 3;
OK
NULL    NULL    NULL    NULL    NULL    NULL    NULL    NULL    NULL
1       12/05/2020      100     1234567.0       electronics     laptop  bangalore       karnataka       manager
2       31/02/2019      101     1234567.0       cloths  top     mangalore       karnataka       manager
Time taken: 0.189 seconds, Fetched: 3 row(s)
hive>
```

10.JOIN query

```
Activities  Terminal                    Fri 13:37
                                        hadoop@bigdata-OptiPlex-360: ~
File  Edit  View  Search  Terminal  Help
OK
Time taken: 0.588 seconds
hive> select * from stu_course;
OK
1       c1      sql     4       dms     f1
2       c1      sql     4       dms     f1
3       c2      os      4       co      f2
4       c2      os      4       co      f2
5       c3      java    4       oops    f3
6       c3      java    4       oops    f3
7       c3      java    4       oops    f3
8       c3      java    4       oops    f3
9       c3      java    4       oops    f3
10      c4      oops    4       c       f2
Time taken: 0.172 seconds, Fetched: 10 row(s)
hive> select st.sid,st.fname,sc.cname from s st join stu_course sc on (st.sid=sc.sid);
Query ID = hadoop_20200710133606_e95a6a85-4ed7-429b-b5cf-66afd6926148
Total jobs = 1

Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1594363444337_0004, Tracking URL = http://bigdata-OptiPlex-360:8088/proxy/application_1594363444337_
0004/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1594363444337_0004
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2020-07-10 13:36:40,197 Stage-3 map = 0%,  reduce = 0%
2020-07-10 13:36:47,521 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 3.05 sec
MapReduce Total cumulative CPU time: 3 seconds 50 msec
Ended Job = job_1594363444337_0004
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 3.05 sec   HDFS Read: 9676 HDFS Write: 203 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 50 msec
OK
1       john    sql
2       mary    sql
3       alex    os
4       tina    os
5       Barney  java
Time taken: 43.154 seconds, Fetched: 5 row(s)
hive>
```

Difference between Internal & External tables:

External Tables -

- External table stores files on the HDFS server but tables are not linked to the source file completely.
- If you delete an external table the file still remains on the HDFS server.

  As an example if you create an **external table** called **"table_test"** in HIVE using HIVE-QL and link the table to file **"file"**, *then deleting "table_test" from HIVE will not delete "file" from HDFS.*
- External table files are accessible to anyone who has access to HDFS file structure and therefore security needs to be managed at the HDFS file/folder level.
- **Meta data is maintained on the master node, and deleting an external table from HIVE only deletes the metadata not the data/file.**

For Internal Tables-

- Stored in a directory based on settings in hive.metastore.warehouse.dir, *by default internal tables are stored* in the following directory *"/user/hive/warehouse"* you can change it by updating the location in the config file .
- Deleting the table deletes the metadata and data from master-node and HDFS respectively.
- Internal table file security is controlled solely via HIVE. Security needs to be managed within HIVE, probably at the schema level (depends on organization).

Hive may have internal or external tables, this is a choice that affects how data is loaded, controlled, and managed.

Use EXTERNAL tables when:

- The **data is also used outside of Hive**. For example, the data files are read and processed by an existing program that doesn't lock the files.
- **Data needs to remain in the underlying location even after a DROP TABLE.** This can apply if you are pointing multiple schema (tables or views) at a single data set or if you are iterating through various possible schema.
- **Hive should not own data and control settings, directories, etc**., you may have another program or process that will do those things.
- **You are not creating table based on existing table (AS SELECT).**

Use INTERNAL tables when:

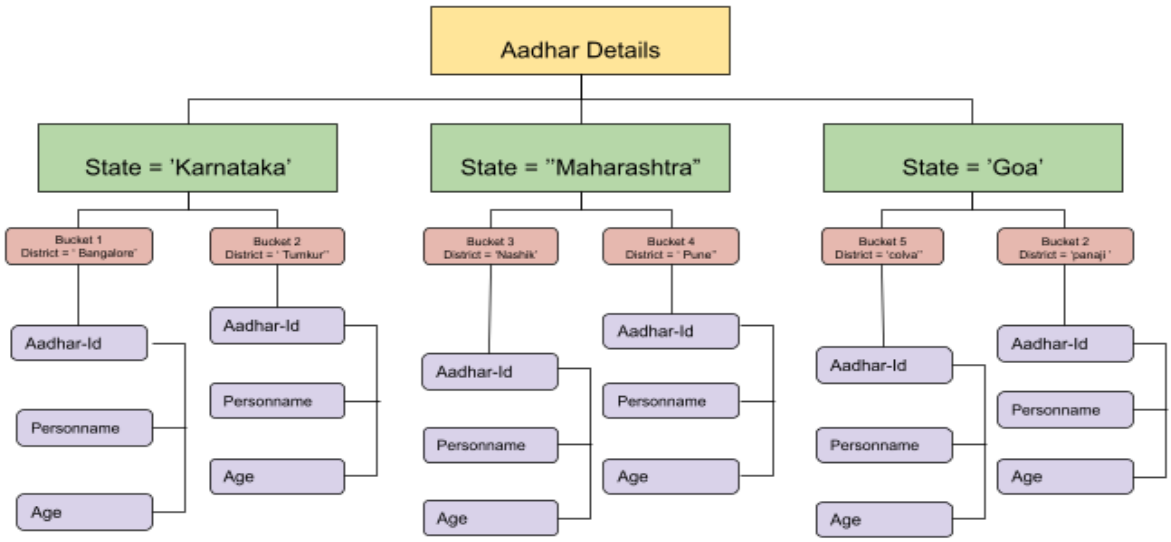
- The **data is temporary**.
- You want **Hive to completely manage the life-cycle of the table and data**

Hive Partitions & Buckets with Example

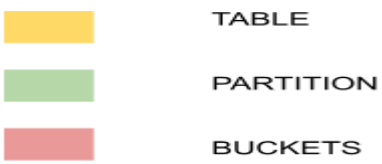
Tables, Partitions, and Buckets are the parts of Hive data modelling.

Apache Hive is an open source data warehouse system used for querying and analysing large datasets.

Partition is helpful when the table has one or more Partition keys. Hive Partitions is a way to organize tables into partitions by dividing tables into different parts based on partition keys that are column's .Partition keys are basic elements for determining how the data is stored in the table.



Note :



- **Partitioning** – organizing tables into partitions for grouping the same type of data together based on a column or partition key. Each table in the hive can have one or more partition keys to identify a particular partition. Using partition we can make it faster to do queries on slices of the data.

  command for Partitioning :

  CREATE TABLE table_name (column1 data_type, column2 data_type)

  PARTITIONED BY (partition1 data_type, partition2 data_type,….);

- **Bucketing** – In Hive Tables or partitions are subdivided into buckets based on the hash function of a column in the table to give extra structure to the data that may be used for more efficient queries.

  command for Bucketing :

  CREATE TABLE table_name PARTITIONED BY (partition1 data_type,

  partition2 data_type,….) CLUSTERED BY (column_name1, column_name2, …)

  SORTED BY (column_name [ASC|DESC], …)] INTO num_buckets BUCKETS;

Advantages and Disadvantages of Hive Partitioning & Bucketing

a) Pros and Cons of Hive Partitioning

Pros:

- It distributes execution load horizontally.
- In partition faster execution of queries with the low volume of data takes place. For example, the search population from Vatican City returns very fast instead of searching the entire world population.

Cons:

- There is the possibility of too many small partition creations- too many directories.
- Partition is effective for low volume data. But there some queries like group by on high volume of data take a long time to execute. For example, the grouping population of China will take a long time as compared to a grouping of the population in Vatican City.
- There is no need for searching the entire table column for a single record.

b) Pros and Cons of Hive Bucketing

Pros:

- It provides faster query responses like portioning.
- In bucketing due to equal volumes of data in each partition, joins at Map side will be quicker.

Cons:

- We can define a number of buckets during table creation. But loading of an equal volume of data has to be done manually by programmers.

So, this was all about Hive Partitioning vs Bucketing.

In conclusion to Hive Partitioning vs Bucketing, we can say that both partition and bucket distributes a subset of the table's data to a subdirectory. Hence, Hive organizes tables into partitions. And it subdivides partitions into buckets.

How will Hive query convert into MapReduce program in the background

Let's understand how and when Hive queries are converted to MapReduce jobs.

First let's have a recap of what map and Reduce means:

- **Map** - Map jobs filter and organise the data in sorted order.
- **Reduce** - Reduce jobs apply summary/aggregate operations across the data.

Depending on the Hive queries, there may be any number of Map and Reduce jobs triggered in the back end.

CASE 1 :

```
describe students;
show tables;
```

These are metadata request queries. In these cases, Hive performs a lookup on the metadata server, which is itself a SQL database (MySQL in most production scenarios).

CASE 2 :

```
select * from students;
```

This is an example of HDFS getting a request. In this case, neither map nor reduce jobs are triggered, since Hive needs to get the data, as is, without applying any computations whatsoever. So Hive executes an equivalent of the *dfs fs -get* command to get the results.

CASE 3 :

```
select id, name from students;
select * from students where fee > 3000;
```

These queries, when executed in Hive, will always trigger some combination of Map and/or Reduce jobs, based on the nature of computation involved. The first 2 queries involve filtering the data (#1 is column wise filter, #2 is row wise filter), hence these will trigger Map only jobs without any Reduce jobs.

CASE 4:

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

The query has only aggregate operation to be applied, which can be done by a Reduce only job. Since no filter or transformation operations are involved, Map job will not be triggered.

CASE 5:

```
select dept, count(*) from students group by dept;
```

The query that involves both Map and Reduce jobs.

This will trigger 1 Map and 1 Reduce job. Map job will do the counting of the students per department, by creating (key, value) pairs for each department where the key is the department name and the value is 1. E.g. - (ISE, 1), (CSE, 1), (ISE, 1), (EC, 1), (EC, 1), etc. The Reduce job will aggregate all of the (key, value) pairs based on the keys to return the final result i.e.

References

[1] https://hadoop.apache.org/

[2] https://data-flair.training/blogs/hadoop-tutorial/

[3] https://www.edureka.co/blog/hadoop-tutorial/

[4] https://www.google.com/