Work Integrated Learning Programmes

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Science & Information Systems

Big Data Systems – Lab Sheet

HIVE

Objectives

Students should be able to

A. Gain understanding about HIVE
B. Process data using various HIVE clauses

Introduction to HIVE

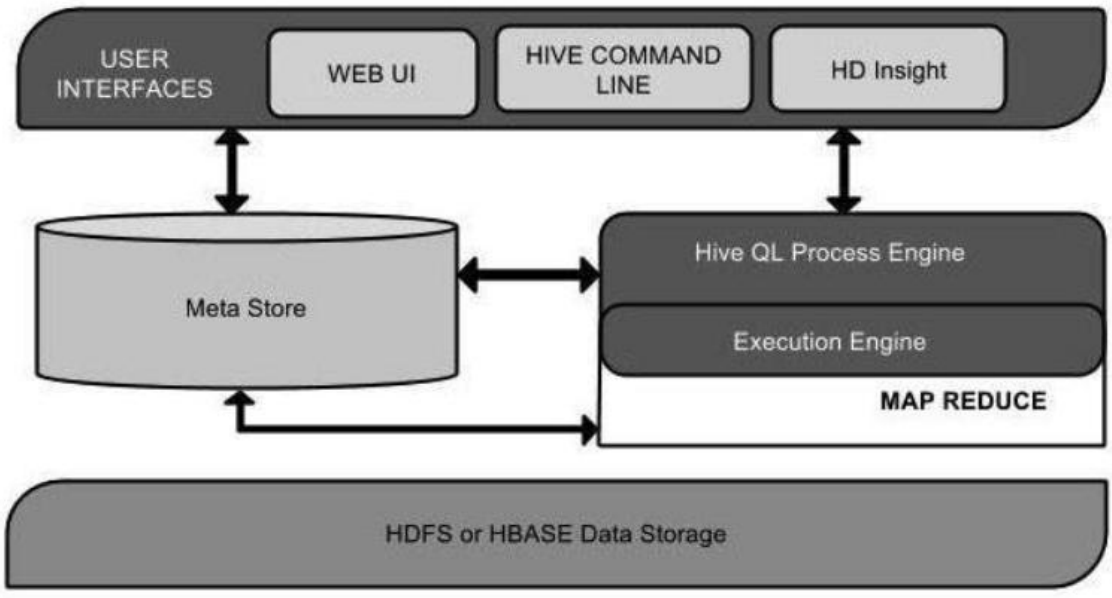
Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analysing easy.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive.

Hive is not designed to be used for OLTP (Online transaction processing) systems rather designed to be used for OLAP (online analytical processing) systems.

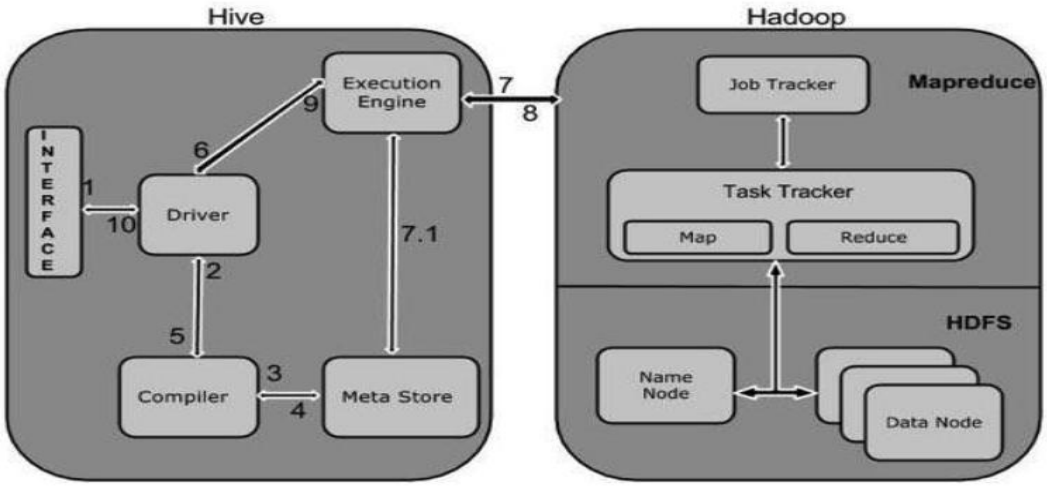
Architecture

The following components depicts the components of hive



Working of HIVE

The following components depict the working of HIVE.

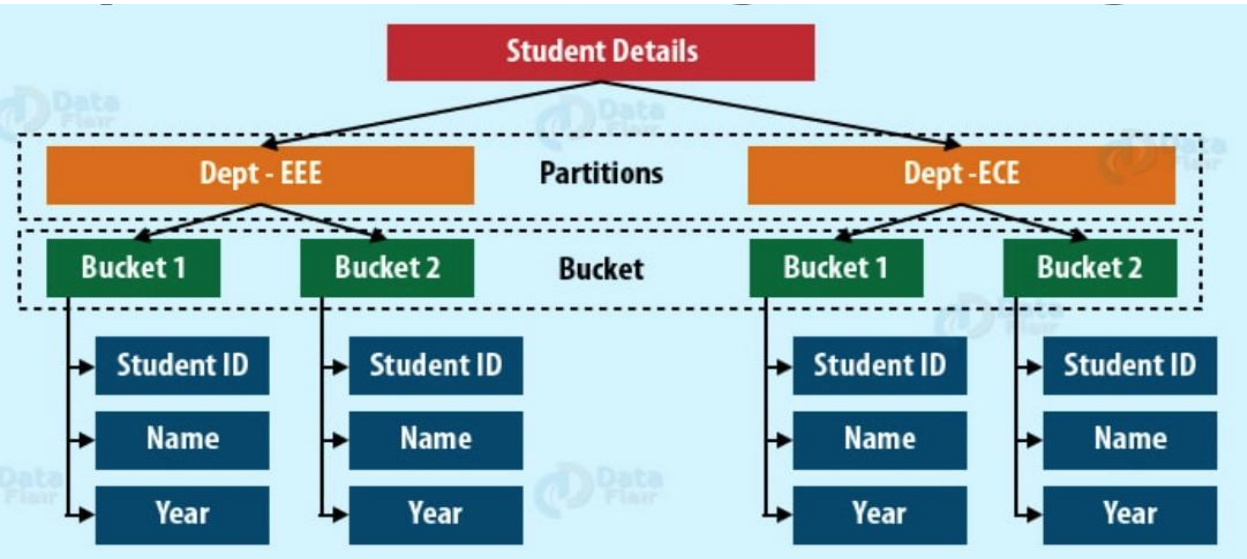


Partitioning and Bucketing in HIVE

Apache Hive allows us to organize the table into multiple partitions where we can group the same kind of data together. It is used for distributing the load horizontally.

When creating a table a key can be used to split data into partitions - implemented as separate sub-dirs with table dir on HDFS.

Bucketing provides Additional level of sub-division within a partition based on hash of some column to make some queries efficient.



HIVE QUERIES

It provides SQL type language for querying called HiveQL or HQL. In this section we will discuss various clauses/operators used within HIVE queries.

In order to access hive shell type hive on terminal as follows

```
[centos@master~]hive
```

CREATE A DATABASE

The create statement is used to create a database. The syntax is as follows

```
hive> create database [IF NOT EXISTS] test;
```

OR

```
hive> create schema [IF NOT EXISTS] test;
```

The above command will create a database with name test. The 'IF NOT EXISTS' clause is optional and will create a database only if the database with name test does not exists already.

SHOW DATABASES

The SHOW databases command is used to list all the databases

```
hive> show databases;
```

The above command will list all the databases.

DROP A DATABASE

The create statement is used to create a database. The syntax is as follows

```
hive> drop database test
```

OR

```
hive> drop schema test;
```

The above command will delete the database test. you can list databases using show databases.

USE DATABASE

In order to create a table you need to go to a particular database first. The USE database command is used to access a database

```
hive> use test;
```

The above command will take you to test1 a database. Now you can create or list tables inside the test database.

CREATE TABLE

The create statement is used to create a table within a database. First go to particular database using '*use database*' command and then type the following command to create a table.

```
hive> create table employee (
employee_id INT,
first_name STRING,
family_name STRING,
    gender STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

The above command will create a table employee with in test database.

SHOW TABLES

The SHOW TABLES command is used to list all the tables with in a database

```
hive> show tables;
```

The above command will list all the tables with in test database.

DESCRIBE

The DESCRIBE command is used to display the table schema

```
hive>desc employee;
```

The above command will display the table schema.

ALTER

The ALTER clause can be used to rename a table, add columns to a table, modify and replace columns

Use of alter to rename a table

```
hive> alter table employee rename to emp;
hive> show tables;
```

Use of alter to add a column

```
hive>alter table emp add columns(address string);
hive>descemp;
```

Use of alter to replace columns

```
hive>alter table emp replace columns(eidint, fname string,
lname string);
hive>descemp;
```

Use of alter to modify a column name

```
hive>alter table emp change fnameename string;
hive>descemp;
```

DROP A TABLE

The drop statement is used to drop a table. The syntax is as follows

```
hive> drop table emp;
hive> show tables;
```

LOAD

The load command is use to load the data from a file into table

Loading data into table from a file stored in local file system

```
hive> load data local inpath 'emp.csv' into table emp;
```

Loading data into table from a file stored in HDFS

```
hive>load data inpath '/hive/input/emp.csv' into table emp;
```

SELECT

The SELECT command is use to retrieve the data from table

Use of SELECT to retrieve all the columns

```
hive> select * from emp;
```

```
10004   'Chirstian'     'Koblick'       'M'
10005   'Kyoichi'       'Maliniak'      'M'
10006   'Anneke'        'Preusig'       'F'
10007   'Tzvetan'       'Zielinski'     'F'
10008   'Saniya'        'Kalloufi'      'M'
```

Use of SELECT to retrieve few columns

```
hive> select employee_id, first_name from emp;
```

```
10004   'Chirstian'
10005   'Kyoichi'
10006   'Anneke'
10007   'Tzvetan'
10008   'Saniya'
```

WHERE

The WHERE clause allows to filter the records based on a condition. The records that satisfy the condition are displayed as result while that do not satisfy the condition are ignored.

Use of WHERE

```
hive>select * from emp where employee_id= 10099;
```

The above command will display all the columns corresponding to employee_id 10099 as follows

```
10099   'Valter'        'Sullins'       'F'
```

Use of WHERE

```
hive> select first_name, gender from emp1 where employee_id=
10099;
```

The above command will display first_name, gender corresponding to employee_id 10099.

```
'Valter'          'F'
```

LIMIT

The LIMIT clause allows to specify a limit on number of records to be displayed as part of result.

Use of LIMIT

```
hive>select * from emp LIMIT 3
```

The above command will display only first 3 records from emp table rather than all the records.

```
10004    'Chirstian'      'Koblick'        'M'
10005    'Kyoichi'        'Maliniak'       'M'
10006    'Anneke'         'Preusig'        'F'
```

ORDER BY

The ORDER by clause is use to sort the output on a particular attribute.

Use of ORDER BY

```
hive>select * from emp order by first_name limit 3
```

The above command will display only first 3 records order by first_name.

```
10060    'Breannda'       'Billingsley'    'M'
10056    'Brendon'        'Bernini'        'F'
10068    'Charlene'       'Brattka'        'M'
```

Use of ORDER BY

```
hive>select employee_id, first_name from emp order by
employee_iddesc limit 5.
```

The above command will display only first 5 records in the descending order of employee_id.

```
10100    'Hironobu'
10099    'Valter'
10098    'Sreekrishna'
10097    'Remzi'
10096    'Jayson'
```

COUNT

The COUNT clause is used to count the number of column values

Use of COUNT

```
hive> select count(*) from emp;
```

The above query will return the number of records in the emp table;

```
hive> select count(*) from emp1;
Query ID = centos_20221217183127_7b148278-8c24-4238-b395-093e384e7832
Total jobs = 1
Launching Job 1 out of 1
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>
2022-12-17 18:31:28,034 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
2022-12-17 18:31:28,054 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
Starting Job = job_1671294842628_0010, Tracking URL = http://master:8088/proxy/application_1671294842628_0010/
Kill Command = /opt/hadoop-3.2.4/bin/mapred job  -kill job_1671294842628_0010
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-17 18:31:35,771 Stage-1 map = 0%,  reduce = 0%
2022-12-17 18:31:41,998 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.75 sec
2022-12-17 18:31:48,229 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.9 sec
MapReduce Total cumulative CPU time: 3 seconds 900 msec
Ended Job = job_1671294842628_0010
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.9 sec   HDFS Read: 15533 HDFS Write: 103 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 900 msec
OK
100
Time taken: 21.542 seconds, Fetched: 1 row(s)
```

MAX

The MAX clause is used find max value from a column

Use of MAX

```
hive> select max(employee_id) from emp;
```

The above query will return the max value of employee_id column.

```
hive> select max(employee_id) from emp1;
Query ID = centos_20221217183435_96b3b544-96e5-401a-bfaa-1b0048987867
Total jobs = 1
Launching Job 1 out of 1
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>
2022-12-17 18:34:35,718 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
2022-12-17 18:34:35,744 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
Starting Job = job_1671294842628_0011, Tracking URL = http://master:8088/proxy/application_1671294842628_0011/
Kill Command = /opt/hadoop-3.2.4/bin/mapred job  -kill job_1671294842628_0011
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-17 18:34:43,419 Stage-1 map = 0%,  reduce = 0%
2022-12-17 18:34:49,575 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2022-12-17 18:34:55,740 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.79 sec
MapReduce Total cumulative CPU time: 4 seconds 790 msec
Ended Job = job_1671294842628_0011
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.79 sec   HDFS Read: 15629 HDFS Write: 105 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 790 msec
OK
10100
Time taken: 21.741 seconds, Fetched: 1 row(s)
```

MIN

The MIN clause is used find minimum value from a column

Use of MIN

```
hive> select min(employee_id) from emp;
```

The above query will return the min value of employee_id column.

```
hive> select min(employee_id) from emp1;
Query ID = centos_20221217183716_fb1bf3d4-8177-46ba-89f9-403ec51fb4e2
Total jobs = 1
Launching Job 1 out of 1
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>
2022-12-17 18:37:17,192 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
2022-12-17 18:37:17,219 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
Starting Job = job_1671294842628_0012, Tracking URL = http://master:8088/proxy/application_1671294842628_0012/
Kill Command = /opt/hadoop-3.2.4/bin/mapred job  -kill job_1671294842628_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-17 18:37:25,009 Stage-1 map = 0%,  reduce = 0%
2022-12-17 18:37:31,174 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.11 sec
2022-12-17 18:37:37,320 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.68 sec
MapReduce Total cumulative CPU time: 4 seconds 680 msec
Ended Job = job_1671294842628_0012
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.68 sec   HDFS Read: 15636 HDFS Write: 105 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 680 msec
OK
10001
Time taken: 21.465 seconds, Fetched: 1 row(s)
```

GROUP BY

The group by is used to group the columns by particular column. It is generally used with statistical functions such as count, min, max ,avg*etc.*

Use of GROUP BY

```
hive> select gender, count(*) from emp group by gender;
```

The above query will return the count of records by gender.

```
Query ID = centos_20221217183956_77b66564-3266-4de5-bdbf-589cda91ca62
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 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>
2022-12-17 18:39:56,244 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
2022-12-17 18:39:56,264 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
Starting Job = job_1671294842628_0013, Tracking URL = http://master:8088/proxy/application_1671294842628_0013/
Kill Command = /opt/hadoop-3.2.4/bin/mapred job  -kill job_1671294842628_0013
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-17 18:40:03,698 Stage-1 map = 0%,  reduce = 0%
2022-12-17 18:40:08,840 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.59 sec
2022-12-17 18:40:17,012 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.0 sec
MapReduce Total cumulative CPU time: 5 seconds 0 msec
Ended Job = job_1671294842628_0013
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.0 sec   HDFS Read: 16176 HDFS Write: 125 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 0 msec
OK
'F'     37
'M'     63
Time taken: 22.061 seconds, Fetched: 2 row(s)
```

Use of GROUP BY

```
hive>select gender, max(employee_id) from emp group by gender;
```

The above query will return the max employee_id from male and female group

```
Query ID = centos_20221217184151_bc4d379a-6a50-4903-8a67-dbfb1711279f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 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>
2022-12-17 18:41:51,352 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
2022-12-17 18:41:51,372 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
Starting Job = job_1671294842628_0014, Tracking URL = http://master:8088/proxy/application_1671294842628_0014/
Kill Command = /opt/hadoop-3.2.4/bin/mapred job  -kill job_1671294842628_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-17 18:41:57,880 Stage-1 map = 0%,  reduce = 0%
2022-12-17 18:42:03,030 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.62 sec
2022-12-17 18:42:09,193 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.27 sec
MapReduce Total cumulative CPU time: 5 seconds 270 msec
Ended Job = job_1671294842628_0014
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.27 sec   HDFS Read: 16082 HDFS Write: 131 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 270 msec
OK
'F'     10100
'M'     10097
Time taken: 19.128 seconds, Fetched: 2 row(s)
```

JOIN

The join operation can be used to join two relations. The joining of two relations is possible in case they have common attribute. There are two types of join.

- Inner join
- Outer join

Inner join

This inner join is used to return the matching rows from 2 relations on a common attribue

Consider the following input files customers.txt (id, name, age, address, salary) and orders.txt (oid, date, customer_id, amount)

Customers.txt

```
1,Ramesh,32,Ahmedabad,2000.00
2,Khilan,25,Delhi,1500.00
3,kaushik,23,Kota,2000.00
4,Chaitali,25,Mumbai,6500.00
5,Hardik,27,Bhopal,8500.00
6,Komal,22,MP,4500.00
7,Muffy,24,Indore,10000.00
```

Orders.txt

```
102,2009-10-08 00:00:00,3,3000
100,2009-10-08 00:00:00,3,1500
101,2009-11-20 00:00:00,2,1560
103,2008-05-20 00:00:00,4,2060
104,2008-05-21 00:00:00,9,2200
```

We can create the customer table as follows

```
hive>create table customers(
    Id INT,
    name STRING,
    age INT,
    address STRING,
    salary double)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

```
hive>load data inpath '/hive/input/customers.txt' into table
customers;
```

The above query will read the file from specified location and load the data into customers table

create orders table

```
hive>create table orders(
oid INT,
o_date DATE,
customer_id INT,
    amount INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

```
hive>load data inpath '/hive/input/orders.txt' into table
orders;
```

The above query will read the file from specified location and load the data into orders table

We can apply join on customers and orders as follows.

```
hive>SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

The result of above join query is shown below

```
54-50_147_7401546440073042 4-1/-local-10004/HashTable-Stage-3/MapJoin-mapfile01--.hashtable
2022-12-17 18:54:58     Uploaded 1 File to: file:/opt/hive-3.1.3/iotmp/868f8faf-393d-46d7-bb8f-ecafa1f0a01b/hive_2022-12-17_18-54-50_147_7401546440073042 4-1/-local-10004
/HashTable-Stage-3/MapJoin-mapfile01--.hashtable (356 bytes)
2022-12-17 18:54:58     End of local task; Time Taken: 1.718 sec.
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
2022-12-17 18:54:59,537 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
2022-12-17 18:54:59,567 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
Starting Job = job_1671294842628_0015, Tracking URL = http://master:8088/proxy/application_1671294842628_0015/
Kill Command = /opt/hadoop-3.2.4/bin/mapred job  -kill job_1671294842628_0015
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2022-12-17 18:55:07,033 Stage-3 map = 0%,  reduce = 0%
2022-12-17 18:55:14,211 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 3.28 sec
MapReduce Total cumulative CPU time: 3 seconds 280 msec
Ended Job = job_1671294842628_0015
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 3.28 sec   HDFS Read: 9886 HDFS Write: 206 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 280 msec
OK
2       Khilan  25      1560
3       kaushik 23      3000
3       kaushik 23      1500
4       Chaital 25      2060
Time taken: 25.132 seconds, Fetched: 4 row(s)
```

Outer Join

outer join returns all the rows (even non matching) from at least one of the relations. An outer join operation is carried out in three ways −

- Left outer join
- Right outer join
- Full outer join

Left outer join

The **left outer Join** operation returns all rows from the left table, even if there are no matches in the right relation. In non-matching rows the attribute values from other tables are filled in with null values.

We can apply join on customers and orders as follows.

```
hive>SELECT c.ID, c.NAME, o.AMOUNT, o.O_DATE
FROM CUSTOMERS c
LEFT OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

The result of above join query is shown below

```
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
2022-12-17 18:59:15,110 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
2022-12-17 18:59:15,130 INFO  [868f8faf-393d-46d7-bb8f-ecafa1f0a01b main] client.RMProxy: Connecting to ResourceManager at master/172.31.1.244:8032
Starting Job = job_1671294842628_0016, Tracking URL = http://master:8088/proxy/application_1671294842628_0016/
Kill Command = /opt/hadoop-3.2.4/bin/mapred job  -kill job_1671294842628_0016
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2022-12-17 18:59:22,718 Stage-3 map = 0%,  reduce = 0%
2022-12-17 18:59:28,934 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 1.75 sec
MapReduce Total cumulative CPU time: 1 seconds 750 msec
Ended Job = job_1671294842628_0016
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 1.75 sec   HDFS Read: 9149 HDFS Write: 344 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 750 msec
OK
1       Ramesh  NULL    NULL
2       Khilan  1560    2009-11-20
3       kaushik 3000    2009-10-08
3       kaushik 1500    2009-10-08
4       Chaital 2060    2008-05-20
5       Hardik  NULL    NULL
6       Komal   NULL    NULL
7       Muffy   NULL    NULL
Time taken: 23.864 seconds, Fetched: 8 row(s)
```

Right outer join

The **right outer Join** operation returns all rows from the right table, even if there are no matches in the left relation. In non-matching rows the attribute values from other tables are filled in with null values.

We can apply join on customers and orders as follows.

```
hive>SELECT c.ID, c.NAME, o.AMOUNT, o.O_DATE
FROM CUSTOMERS c
RIGHT OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

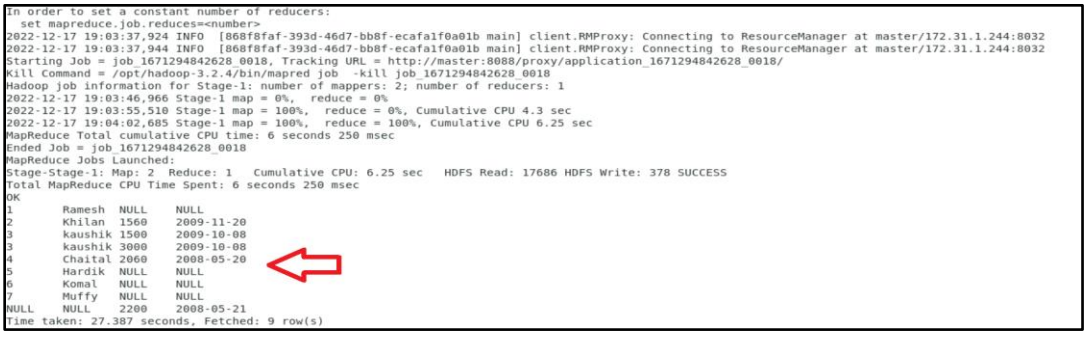
The result of above join query is shown below

Full outer join

The **full outer Join** operation returns matching and non-matching rows from both the tables. In case of non-matching rows the attributes from other table are filled in with null values.
We can apply join on customers and orders as follows.

```
hive>SELECT c.ID, c.NAME, o.AMOUNT, o.O_DATE
FROM CUSTOMERS c
FULL OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

The result of above join qury is shown below



Partitioning

Hive table partition is a way to split a large table into smaller tables based on one or more partition keys.

Use of partition in HIVE

```
hive> CREATE TABLE zipcodes(
RecordNumberint,
Zipcodeint,
City string,
State string
)
PARTITIONED BY(Country string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

The above query will create the partitions by country attribute. Further we will load the data from the input file into this partitioned table

Input file: zipcodes.csv

```
RecordNumber,Zipcode,City,State,Country
1,151203,'FDK','PB','India'
2,151204,'KKP','PB','India'
3,151205,'BTI','PB','India'
4,151206,'JAL','PB','India'
5,151207,'LDH','PB','India'
6,251204,'GGN','RJ','India'
7,251205,'JPR','RJ','India'
8,251206,'BKR','RJ','India'
9,251207,'JOR','RJ','India'
10,251208,'UDP','RJ','India'
11,351203,'AMB','HR','India'
12,351204,'KUK','HR','India'
13,351205,'KAR','HR','India'
14,351205,'PPT','HR','India'
15,351207,'SPT','HR','India'
16,651203,'LA','CA','USA'
17,651204,'SD','CA','USA'
18,651205,'SJ','CA','USA'
19,651205,'SF','CA','USA'
20,651207,'LB','CA','USA'
```

Load the data file into the table using following command.

```
hive>load data inpath '/hive/input/zipcodes.csv' into table
zipcodes;
```

When you load the data into the partition table, Hive internally splits the records based on the partition key and stores each partition data into a sub-directory of tables directory on HDFS.

You can use the following command to look at the directory structure

```
[centos@master~]$hadoop fs -ls
/user/hive/warehouse/test.db/zipcodes/
```

```
Found 2 items
drwxr-xr-x   - centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27India%27
drwxr-xr-x   - centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27USA%27
```

Bucketing

Hive Bucketing is a way to split the table into a managed number of clusters with or without partitions

Use of bucketing in HIVE

```
hive>CREATE TABLE zipcodes(
RecordNumberint,
Zipcodeint,
City string,
State string,
Country string)
CLUSTERED BY (State) INTO 4 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

The above query will create 4 buckets by state.

Load the data file into the table using following command.

```
hive>load data inpath '/hive/input/zipcodes.csv' into table
zipcodes;
```

When you load the data into the partition table, Hive internally splits the records based on the partition key and stores each partition data into a sub-directory of tables directory on HDFS.

Each bucket is stored as a file within the table's directory. You can use the following command to see the directory structure.

```
[centos@master~]$hadoop fs -ls
/user/hive/warehouse/test.db/zipcodes/
```

```
-rw-r--r--   3 centos hadoop        174 2022-12-17 16:47 /user/hive/warehouse/test1.db/zipcodes5/000000_0
-rw-r--r--   3 centos hadoop          0 2022-12-17 16:47 /user/hive/warehouse/test1.db/zipcodes5/000001_0
-rw-r--r--   3 centos hadoop        382 2022-12-17 16:47 /user/hive/warehouse/test1.db/zipcodes5/000002_0
-rw-r--r--   3 centos hadoop          0 2022-12-17 16:47 /user/hive/warehouse/test1.db/zipcodes5/000003_0
```

You can also create bucketing on a partitioned table to further split the data to improve the query performance of the partitioned table.

Use of partitioning and bucketing in HIVE

```
hive>CREATE TABLE zipcodes4(
RecordNumberint,
Zipcodeint,
City string,
State string
)
PARTITIONED BY(country string)
CLUSTERED BY (State) INTO 4 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

The above query will create partitioning based on country then with in each partition it will create 4 buckets based on state.

You can use the following commands to list the directory structure.

```
[centos@master~]$hadoop fs -ls
/user/hive/warehouse/test.db/zipcodes/
```

```
Found 2 items
drwxr-xr-x   - centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27India%27
drwxr-xr-x   - centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27USA%27
```

Further you can list a partitioned directory as follows.

```
[centos@master~]$hadoop fs -ls
/user/hive/warehouse/test.db/zipcodes/country=%27India%27
```

```
-rw-r--r--   3 centos hadoop        126 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27India%27/000000_0
-rw-r--r--   3 centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27India%27/000001_0
-rw-r--r--   3 centos hadoop        180 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27India%27/000002_0
-rw-r--r--   3 centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27India%27/000003_0
```

```
[centos@master~]$hadoop fs -ls
/user/hive/warehouse/test.db/zipcodes/country=%27USA%27
```

```
-rw-r--r--   3 centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27USA%27/000000_0
-rw-r--r--   3 centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27USA%27/000001_0
-rw-r--r--   3 centos hadoop        100 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27USA%27/000002_0
-rw-r--r--   3 centos hadoop          0 2022-12-17 16:53 /user/hive/warehouse/test1.db/zipcodes4/country=%27USA%27/000003_0
```

You can query partitioned and bucketed table as you query other tables. The queries that involve condition on partitioned or bucketed attributes will give better performance on large datasets.

CREATING EXTERNAL TABLE

Hive owns the data for the internal tables. By default, an internal table will be created in a folder path similar to **/user/hive/warehouse** directory of HDFS. If we drop the managed table or partition, the table data and the metadata associated with that table will be deleted from the HDFS.

Hive does not manage the data of the External tables. External tables are stored outside the warehouse directory. Whenever we drop the external table, then only the metadata associated with the table will get deleted, the table data remains untouched by Hive.

We can create the external table by specifying the **EXTERNAL** keyword in the Hive create table statement.

```
hive>CREATE external TABLE zipcodes_ext(
RecordNumberint,
Zipcodeint,
City string,
State string,
Country string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
Location '/hive/ext_table'; //location where the data will
                            //reside
```

The above command will create an external table zipcodes_ext and the data for the same will be stored in '/hive/ext_table'

Load the data file into the table using following command.

```
hive>load data inpath '/hive/input/zipcodes.csv' into table
zipcodes_ext;
```

you can see this input data file in the specified directory as follows.

```
[centos@master~]$hadoop fs -ls /hive/ext_table
```

Outputs/Results

- Students should be able to appreciate the usage of HIVE queries.
- Students should be able to appreciate partitioning and bucketing feature of HIVE

Observations

Students should carefully observe the syntax of HIVE queries and verify the output

References

- Tutorial point
- Edureka
- Spark apache hive
- Analytics Vidya
- Data Flair