Problem Statement:

Explain the below concepts with an example in brief.

1. Hive Data Definitions

Solutions:

Data Definition Language (DDL)

DDL statements are used to build and modify the tables and other objects in the database.

Example :

CREATE, DROP, TRUNCATE, ALTER, SHOW, DESCRIBE Statements.

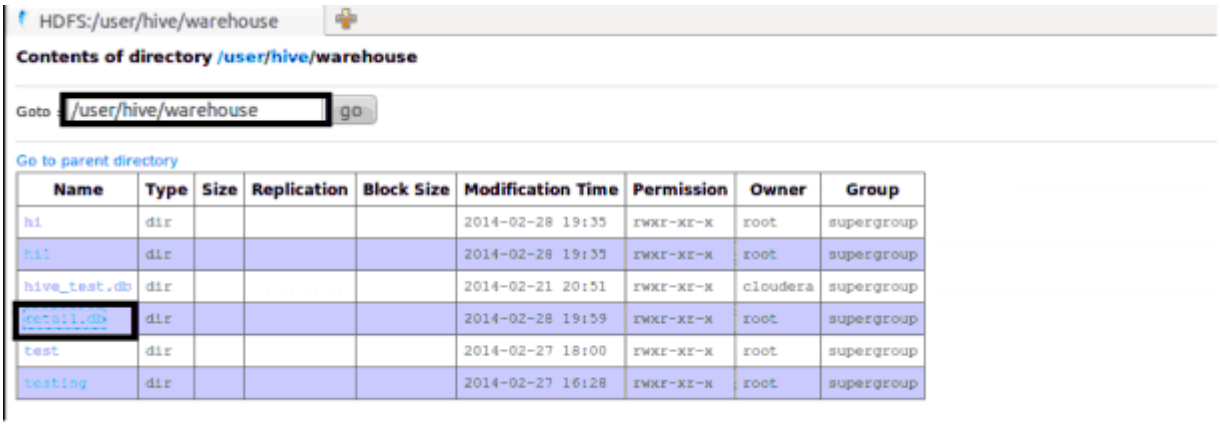
Go to Hive shell by giving the command sudo hive and enter the command **'create database<data base name>'** to create the new database in the Hive.

```
hive> create database retail;
OK
Time taken: 5.275 seconds
hive>
```

To list out the databases in Hive warehouse, enter the command '**show databases'.**

```
hive> show databases;
OK
default
retail
Time taken: 0.228 seconds
hive>
```

The database creates in a default location of the Hive warehouse. In Cloudera, Hive database store in a /user/hive/warehouse.



Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
hi	dir				2014-02-28 19:35	rwxr-xr-x	root	supergroup
hi1	dir				2014-02-28 19:35	rwxr-xr-x	root	supergroup
hive_test.db	dir				2014-02-21 20:51	rwxr-xr-x	cloudera	supergroup
retail.db	dir				2014-02-28 19:59	rwxr-xr-x	root	supergroup
test	dir				2014-02-27 18:00	rwxr-xr-x	root	supergroup
testing	dir				2014-02-27 16:28	rwxr-xr-x	root	supergroup

The command to use the database is **USE <data base name>**

```
hive> use retail;
OK
Time taken: 0.023 seconds
hive>
```

Copy the input data to HDFS from local by using the copy From Local command.

txns1.txt

```
00000000,06-26-2011,4007024,040.33,Exercise & Fitness,Cardio Machine Accessories,Clarksville,Tennessee,credit
00000001,05-26-2011,4006742,198.44,Exercise & Fitness,Weightlifting Gloves,Long Beach,California,credit
00000002,06-01-2011,4009775,005.58,Exercise & Fitness,Weightlifting Machine Accessories,Anaheim,California,credit
00000003,06-05-2011,4002199,198.19,Gymnastics,Gymnastics Rings,Milwaukee,Wisconsin,credit
00000004,12-17-2011,4002613,098.81,Team Sports,Field Hockey,Nashville  ,Tennessee,credit
00000005,02-14-2011,4007591,193.63,Outdoor Recreation,Camping & Backpacking & Hiking,Chicago,Illinois,credit
00000006,10-28-2011,4002190,027.89,Puzzles,Jigsaw Puzzles,Charleston,South Carolina,credit
00000007,07-14-2011,4002964,096.01,Outdoor Play Equipment,Sandboxes,Columbus,Ohio,credit
00000008,01-17-2011,4007361,010.44,Winter Sports,Snowmobiling,Des Moines,Iowa,credit
00000009,05-17-2011,4004798,152.46,Jumping,Bungee Jumping,St. Petersburg,Florida,credit
```

```
cloudera@cloudera-vm:~$ hadoop dfs -copyFromLocal Desktop/blog/txns1.txt hdfs:/
cloudera@cloudera-vm:~$
```

When we create a table in hive, it creates in the default location of the hive warehouse. – "/user/hive/warehouse", after creation of the table we can move the data from HDFS to hive table.

The following command creates a table with in location of "/user/hive/warehouse/retail.db"

Note : retail.db is the database created in the Hive warehouse.

```
hive> create table txnrecords(txnno INT, txndate STRING, custno INT, amount DOUBLE,category STRING, product STRING, city STRIN
G, state STRING, spendby STRING) row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 1.163 seconds
hive>
```

Describe provides information about the schema of the table.

```
hive> describe txnrecords;
OK
txnno   int
txndate string
custno  int
amount  double
category        string
product string
city    string
state   string
spendby string
Time taken: 0.122 seconds
hive>
```

2. Hive Data Manipulations

Solutions:

Data Manipulation Language (DML)

DML statements are used to retrieve, store, modify, delete, insert and update data in the database.

Example :

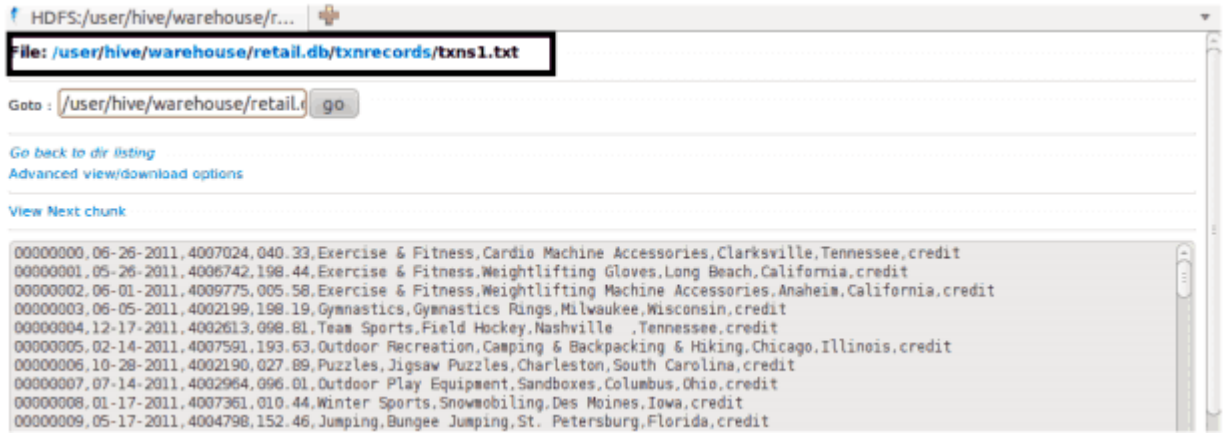
LOAD, INSERT Statements.

Syntax :

LOAD data <LOCAL> inpath <file path> into table [tablename]

The Load operation is used to move the data into corresponding Hive table. If the keyword **local** is specified, then in the load command will give the local file system path. If the keyword local is not specified we have to use the HDFS path of the file.

```
hive> LOAD DATA INPATH '/txns1.txt' OVERWRITE INTO TABLE txnrecords;
Loading data to table retail.txnrecords
Deleted hdfs://localhost/user/hive/warehouse/retail.db/txnrecords
OK
Time taken: 0.263 seconds
hive>
```



Here are some examples for the LOAD data LOCAL command

```
hive> create table customer(custno string, firstname string, lastname string, age int,profession string) row format delimited
fields terminated by ',';
OK
Time taken: 0.102 seconds
hive>
```

```
hive> load data local inpath '/home/cloudera/Desktop/blog/custs' into table customer;
Copying data from file:/home/cloudera/Desktop/blog/custs
Copying file: file:/home/cloudera/Desktop/blog/custs
Loading data to table retail.customer
OK
Time taken: 0.227 seconds
hive>
```

After loading the data into the Hive table we can apply the Data Manipulation Statements or aggregate functions retrieve the data.

Example to count number of records:

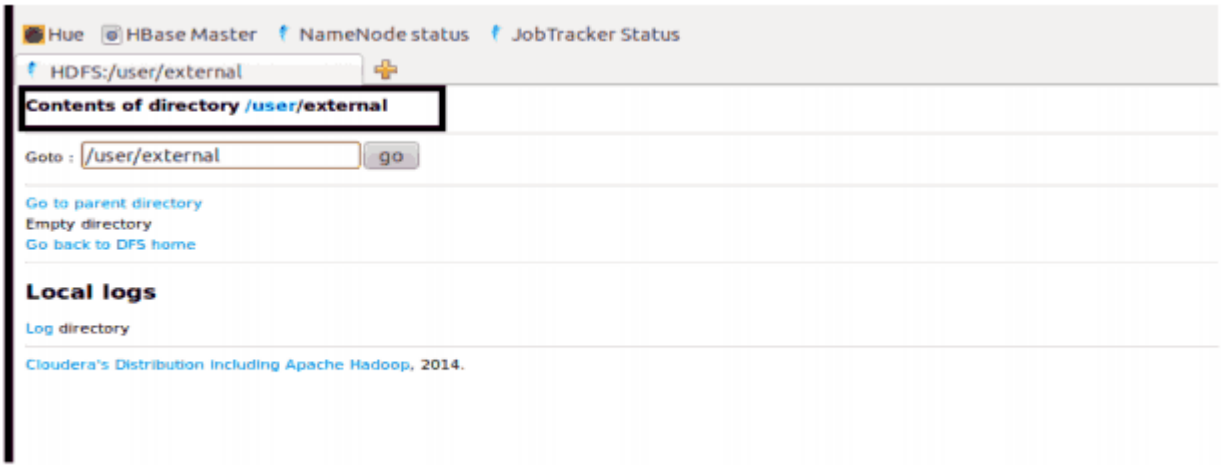
Count aggregate function is used count the total number of the records in a table.

```
hive> select count(*) from txnrecords;
Total MapReduce 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 mapred.reduce.tasks=<number>
Starting Job = job_201402270420_0005, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201402270420_0005
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201402270420_0005
2014-02-28 20:02:41,231 Stage-1 map = 0%,  reduce = 0%
2014-02-28 20:02:48,293 Stage-1 map = 50%,  reduce = 0%
2014-02-28 20:02:49,309 Stage-1 map = 100%,  reduce = 0%
2014-02-28 20:02:55,350 Stage-1 map = 100%,  reduce = 33%
2014-02-28 20:02:56,367 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201402270420_0005
OK
50000
Time taken: 19.027 seconds
hive>
```

'create external' Table :

The **create external** keyword is used to create a table and provides a location where the table will create, so that Hive does not use a default location for this table. An **EXTERNAL** table points to any HDFS location for its storage, rather than default storage.

```
hive> create external table example_customer(custno string, firstname string, lastname string, age int,profession string) row
format delimited fields terminated by ',' LOCATION '/user/external';
OK
Time taken: 0.059 seconds
hive>
```



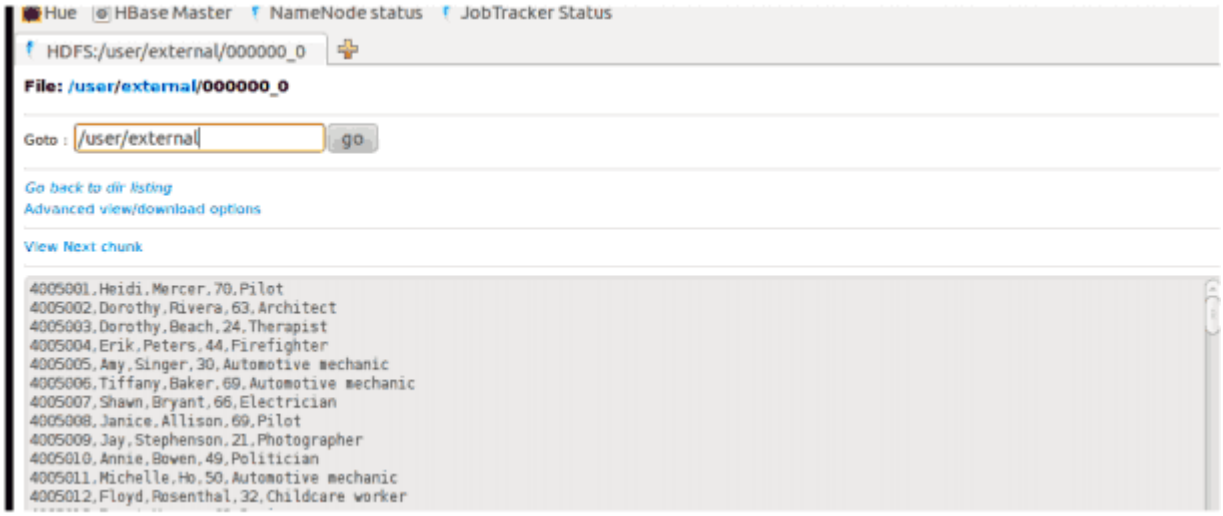
Insert Command:

The **insert** command is used to load the data Hive table. Inserts can be done to a table or a partition.

• INSERT OVERWRITE is used to overwrite the existing data in the table or partition.

• INSERT INTO is used to append the data into existing data in a table. (Note: INSERT INTO syntax is work from the version 0.8)

```
hive> from customer cus insert overwrite table example_customer select cus.custno,cus.firstname,cus.lastname,cus.age,cus.profe
ssion;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201402270420_0007, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201402270420_0007
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201402270420_0007
2014-02-28 20:40:39,866 Stage-1 map = 0%,  reduce = 0%
2014-02-28 20:40:41,871 Stage-1 map = 100%,  reduce = 0%
2014-02-28 20:40:42,876 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201402270420_0007
Loading data to table retail.example_customer
Deleted hdfs://localhost/user/external
Table retail.example_customer stats: [num_partitions: 0, num_files: 0, num_rows: 0, total_size: 0]
9999 Rows loaded to example_customer
OK
Time taken: 5.786 seconds
hive>
```



Example for 'Partitioned By' and 'Clustered By' Command :

'**Partitioned by**' is used to divided the table into the Partition and can be divided in to buckets by using the '**Clustered By**' command.

```
hive> create table txnrecsByCat(txnno INT, txndate STRING, custno INT, amount DOUBLE,product STRING, city STRING, state STRING
, spendby STRING) partitioned by (category STRING) clustered by (state) INTO 10 buckets row format delimited fields terminated
 by ',' stored as textfile;
OK
Time taken: 0.101 seconds
hive>
```

```
hive> from txnrecords txn INSERT OVERWRITE TABLE record PARTITION(category)select txn.txnno,txn.txndate,txn.custno,txn.amount,
txn.product,txn.city,txn.state,txn.spendby, txn.category;
FAILED: Error in semantic analysis: Dynamic partition strict mode requires at least one static partition column. To turn this
off set hive.exec.dynamic.partition.mode=nonstrict
```

When we insert the data Hive throwing errors, the dynamic partition mode is strict and dynamic partition not enabled (by Jeff at dresshead website). So we need to set the following parameters in Hive shell.

set hive.exec.dynamic.partition=true;

To enable dynamic partitions, by default, it’s false

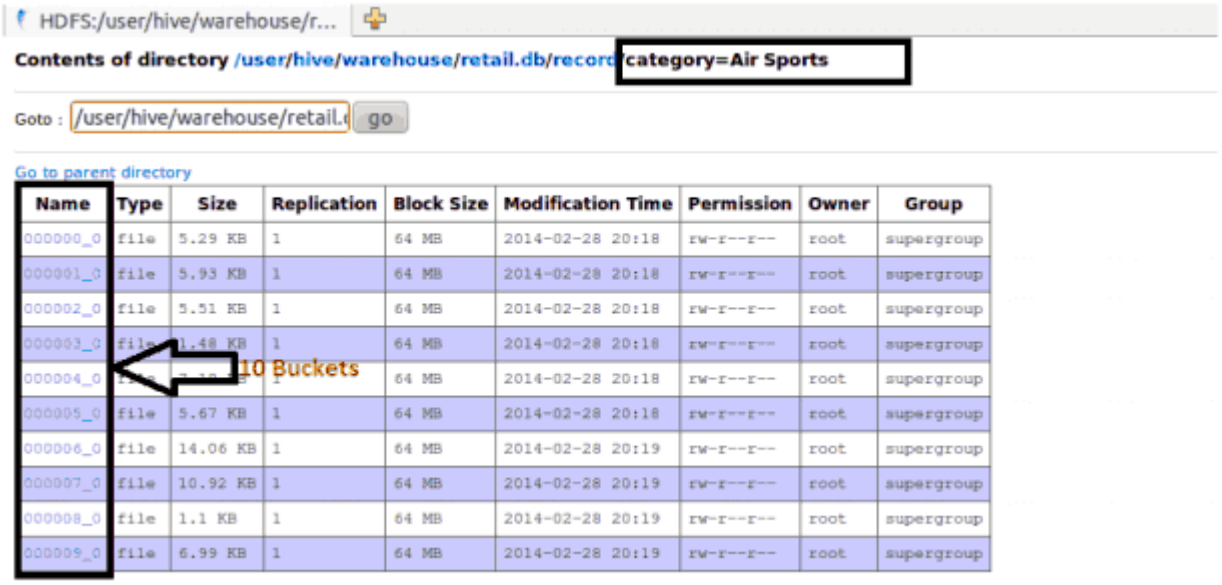
set hive.exec.dynamic.partition.mode=nonstrict;

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

```
hive> set hive.exec.dynamic.partition=true;
```

```
hive> from txnrecords txn INSERT OVERWRITE TABLE record PARTITION(category)select txn.txnno,txn.txndate,txn.custno,txn.amount,
txn.product,txn.city,txn.state,txn.spendby, txn.category;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 10
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 mapred.reduce.tasks=<number>
Starting Job = job_201402270420_0006, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201402270420_0006
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201402270420_0006
2014-02-28 20:18:22,243 Stage-1 map = 0%,  reduce = 0%
2014-02-28 20:18:29,289 Stage-1 map = 100%,  reduce = 0%
2014-02-28 20:18:39,352 Stage-1 map = 100%,  reduce = 10%
2014-02-28 20:18:40,360 Stage-1 map = 100%,  reduce = 20%
2014-02-28 20:18:49,412 Stage-1 map = 100%,  reduce = 40%
2014-02-28 20:18:58,456 Stage-1 map = 100%,  reduce = 50%
2014-02-28 20:18:59,459 Stage-1 map = 100%,  reduce = 60%
2014-02-28 20:19:09,506 Stage-1 map = 100%,  reduce = 80%
2014-02-28 20:19:18,547 Stage-1 map = 100%,  reduce = 90%
2014-02-28 20:19:19,554 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201402270420_0006
```

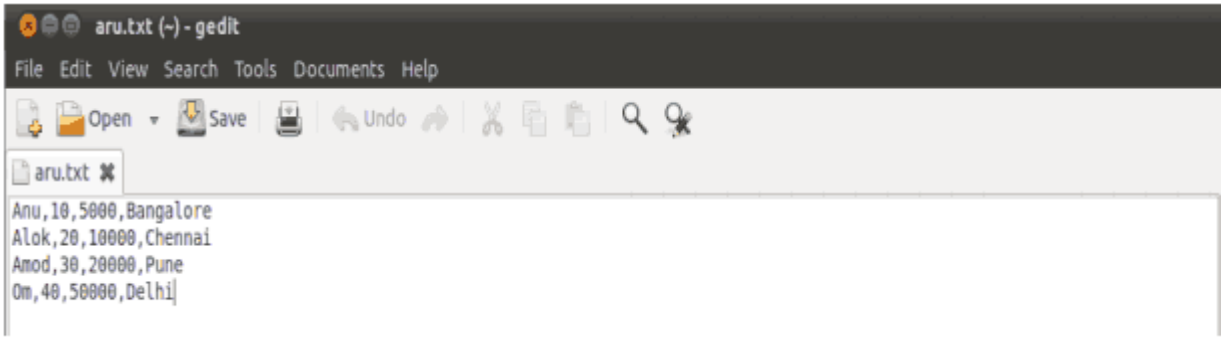
Partition is done by the category and can be divided in to buckets by using the ‘Clustered By’ command.



Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
000000_0	file	5.29 KB	1	64 MB	2014-02-28 20:18	rw-r--r--	root	supergroup
000001_0	file	5.93 KB	1	64 MB	2014-02-28 20:18	rw-r--r--	root	supergroup
000002_0	file	5.51 KB	1	64 MB	2014-02-28 20:18	rw-r--r--	root	supergroup
000003_0	file	1.48 KB	1	64 MB	2014-02-28 20:18	rw-r--r--	root	supergroup
000004_0	file	5.78 KB	1	64 MB	2014-02-28 20:18	rw-r--r--	root	supergroup
000005_0	file	5.67 KB	1	64 MB	2014-02-28 20:18	rw-r--r--	root	supergroup
000006_0	file	14.06 KB	1	64 MB	2014-02-28 20:19	rw-r--r--	root	supergroup
000007_0	file	10.92 KB	1	64 MB	2014-02-28 20:19	rw-r--r--	root	supergroup
000008_0	file	1.1 KB	1	64 MB	2014-02-28 20:19	rw-r--r--	root	supergroup
000009_0	file	6.99 KB	1	64 MB	2014-02-28 20:19	rw-r--r--	root	supergroup

The ‘Drop Table’ statement deletes the data and metadata for a table. In the case of external tables, only the metadata is deleted.

```
hive> drop table customer;
OK
Time taken: 0.922 seconds
```



The 'Drop Table' statement deletes the data and metadata for a table. In the case of external tables, only the metadata is deleted.

Load data local inpath 'aru.txt' into table tablename and then we check employee1 table by using Select * from table name command

```
hive> load data local inpath 'aru.txt' into table employee1;
Copying data from file:/home/cloudera/aru.txt
Copying file: file:/home/cloudera/aru.txt
Loading data to table arushi.employee1
OK
Time taken: 0.434 seconds
hive> select * from employee1;
OK
Anu     10      5000.0  Bangalore
Alok    20      10000.0 Chennai
Amod    30      20000.0 Pune
Om      40      50000.0 Delhi
Time taken: 0.213 seconds
```

To count the number of records in table by using Select **count(*)** from txnrecords;

```
hive> select count(*) from employee1;
Total MapReduce 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 mapred.reduce.tasks=<number>
Starting Job = job_201312102209_0008, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0008
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0008
2013-12-11 00:58:36,125 Stage-1 map = 0%,  reduce = 0%
2013-12-11 00:58:39,154 Stage-1 map = 100%,  reduce = 0%
2013-12-11 00:58:46,204 Stage-1 map = 100%,  reduce = 33%
2013-12-11 00:58:47,214 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0008
OK
4
Time taken: 14.897 seconds
```

Aggregation :

Select count (DISTINCT category) from tablename;

This command will count the different category of ‘cate’ table. Here there are 3 different categories.

Suppose there is another table cate where f1 is field name of category.

```
hive> select * from cate;
OK
category1       1000
category2       200
category1       1000
category3       5000
category2       200
category1       1000
category2       200
category1       1000
category2       200
category3       5000
Time taken: 0.219 seconds
```

```
hive> select count(distinct f1) from cate;
Total MapReduce 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 mapred.reduce.tasks=<number>
Starting Job = job_201312102209_0010, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0010
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0010
2013-12-11 01:04:07,180 Stage-1 map = 0%,  reduce = 0%
2013-12-11 01:04:09,190 Stage-1 map = 100%,  reduce = 0%
2013-12-11 01:04:16,224 Stage-1 map = 100%,  reduce = 33%
2013-12-11 01:04:17,231 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0010
OK
3
Time taken: 13.577 seconds
```

Grouping :

Group command is used to group the result-set by one or more columns.

Select category, sum(amount) from txt records group by category

It calculates the amount of same category.

```
hive> select f1, sum(f2) from cate group by f1;
Total MapReduce 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 mapred.reduce.tasks=<number>
Starting Job = job_201312102209_0011, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0011
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0011
2013-12-11 01:05:36,284 Stage-1 map = 0%,  reduce = 0%
2013-12-11 01:05:38,292 Stage-1 map = 100%,  reduce = 0%
2013-12-11 01:05:45,326 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0011
OK
category1       4000
category2       800
category3       10000
Time taken: 12.453 seconds
```

The result one table is stored in to another table.

Create table newtablename as select * from oldtablename;

```
hive> create table result as select * from cate;
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201312102209_0012, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0012
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0012
2013-12-11 01:09:44,943 Stage-1 map = 0%,  reduce = 0%
2013-12-11 01:09:46,957 Stage-1 map = 100%,  reduce = 0%
2013-12-11 01:09:47,970 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0012
Ended Job = 20115431, job is filtered out (removed at runtime).
Launching Job 2 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201312102209_0013, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0013
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0013
2013-12-11 01:09:50,216 Stage-2 map = 0%,  reduce = 0%
2013-12-11 01:09:51,224 Stage-2 map = 100%,  reduce = 0%
2013-12-11 01:09:52,230 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0013
```

Join Command :

Here one more table is created in the name **‘mailid’**

```
hive> create table mailid(name string, email string)
    > row format delimited
    > fields terminated by ',';
OK
Time taken: 0.081 seconds
```

```
hive> select * from mailid;
OK
anu     anu@gmail.com
om      om@yahoo.com
Anu     anu@gmail.com
Om      om@yahoo.com
Alok    alok@gmail.com
Time taken: 0.126 seconds
```

Join Operation:

A Join operation is performed to combining fields from two tables by using values common to each.

```
hive> select a.name,a.age,a.salary,b.email from employee1 a
    > join mailid b on a.name = b.name;
Total MapReduce 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 mapred.reduce.tasks=<number>
Starting Job = job_201312102209_0017, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0017
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0017
2013-12-11 01:30:34,669 Stage-1 map = 0%,  reduce = 0%
2013-12-11 01:30:36,677 Stage-1 map = 67%,  reduce = 0%
2013-12-11 01:30:38,688 Stage-1 map = 100%,  reduce = 0%
2013-12-11 01:30:44,721 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0017
OK
Alok    20      10000.0 alok@gmail.com
Anu     10      5000.0  anu@gmail.com
Om      40      50000.0 om@yahoo.com
```

Left Outer Join:

The result of a left outer join (or simply left join) for tables A and B always contains all records of the “left” table (A), even if the join-condition does not find any matching record in the “right” table (B).

```
hive> select a.name,a.age,a.salary,b.email from employee1 a
    > left outer join mailid b on a.name = b.name;
```

```
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 mapred.reduce.tasks=<number>
Starting Job = job_201312102209_0018, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0018
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0018
2013-12-11 01:35:13,880 Stage-1 map = 0%,  reduce = 0%
2013-12-11 01:35:15,887 Stage-1 map = 67%,  reduce = 0%
2013-12-11 01:35:17,897 Stage-1 map = 100%,  reduce = 0%
2013-12-11 01:35:22,920 Stage-1 map = 100%,  reduce = 33%
2013-12-11 01:35:23,926 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0018
OK
Alok    20      10000.0 alok@gmail.com
Amod    30      20000.0 NULL
Anu     10      5000.0  anu@gmail.com
Om      40      50000.0 om@yahoo.com
Time taken: 13.464 seconds
```

Right Outer Join:

A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the “right” table (B) will appear in the joined table at least once.

```
hive> select a.name,a.age,a.salary,b.email from employee1 a
    > right outer join mailid b on a.name = b.name;
```

```
Total MapReduce 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 mapred.reduce.tasks=<number>
Starting Job = job_201312102209_0019, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0019
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0019
2013-12-11 01:37:53,768 Stage-1 map = 0%,  reduce = 0%
2013-12-11 01:37:55,775 Stage-1 map = 67%,  reduce = 0%
2013-12-11 01:37:57,789 Stage-1 map = 100%,  reduce = 0%
2013-12-11 01:38:03,817 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0019
OK
Alok    20      10000.0 alok@gmail.com
Anu     10      5000.0  anu@gmail.com
Om      40      50000.0 om@yahoo.com
NULL    NULL    NULL    anu@gmail.com
NULL    NULL    NULL    om@yahoo.com
```

Full Join:

The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

```
hive> select a.name,a.age,a.salary,b.email from employee1 a
    > full join mailid b on a.name = b.name;
```

```
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 mapred.reduce.tasks=<number>
Starting Job = job_201312102209_0020, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201312102209_0020
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:80
21 -kill job_201312102209_0020
2013-12-11 01:40:18,206 Stage-1 map = 0%,  reduce = 0%
2013-12-11 01:40:20,213 Stage-1 map = 67%,  reduce = 0%
2013-12-11 01:40:22,222 Stage-1 map = 100%,  reduce = 0%
2013-12-11 01:40:28,251 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201312102209_0020
OK
Alok    20      10000.0 alok@gmail.com
Amod    30      20000.0 NULL
Anu     10      5000.0  anu@gmail.com
Om      40      50000.0 om@yahoo.com
NULL    NULL    NULL    anu@gmail.com
NULL    NULL    NULL    om@yahoo.com
```

Once done with hive we can use quit command to exit from the hive shell.

```
hive> quit;
```

3. HiveQL Manipulations

Solutions:

The Hive Query Language (HiveQL) is a query language for Hive to process and analyze structured data in a Metastore.

HiveQL - Select-Where

SELECT statement is used to retrieve the data from a table. WHERE clause works similar to a condition. It filters the data using the condition and gives you a finite result. The built-in operators and functions generate an expression, which fulfils the condition.

Syntax

Given below is the syntax of the SELECT query:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference

[WHERE where_condition]

[GROUP BY col_list]

[HAVING having_condition]

[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]

[LIMIT number];
```

Example

Assume we have the employee table as given below, with fields named Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000.

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

The following query retrieves the employee details using the above scenario:

```
hive> SELECT * FROM employee WHERE salary>30000;
```

On successful execution of the query, you get to see the following response:

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
+------+--------------+-------------+-------------------+--------+
```

JDBC Program

The JDBC program to apply where clause for the given example is as follows.

```
import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;



public class HiveQLWhere {

   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
```

```
   public static void main(String[] args) throws SQLException {


      // Register driver and create driver instance

      Class.forName(driverName);


      // get connection

      Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");


      // create statement

      Statement stmt = con.createStatement();


      // execute statement

      Resultset res = stmt.executeQuery("SELECT * FROM employee WHERE salary>30000;");


      System.out.println("Result:");

      System.out.println(" ID \t Name \t Salary \t Designation \t Dept ");


      while (res.next()) {

         System.out.println(res.getInt(1) + " " + res.getString(2) + " " + res.getDouble(3) + " " +
res.getString(4) + " " + res.getString(5));

      }

      con.close();

   }

}
```

Save the program in a file named HiveQLWhere.java. Use the following commands to compile and execute this program.

```
$ javac HiveQLWhere.java

$ java HiveQLWhere
```

Output:

```
ID       Name           Salary      Designation          Dept
1201     Gopal          45000       Technical manager    TP
1202     Manisha        45000       Proofreader          PR
1203     Masthanvali    40000       Technical writer     TP
1204     Krian          40000       Hr Admin             HR
```

HiveQL - Select-Order By

The ORDER BY clause is used to retrieve the details based on one column and sort the result set by ascending or descending order.

Syntax

Given below is the syntax of the ORDER BY clause:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference

[WHERE where_condition]

[GROUP BY col_list]

[HAVING having_condition]

[ORDER BY col_list]]

[LIMIT number];
```

Example

The following query retrieves the employee details using the above scenario:

```
hive> SELECT Id, Name, Dept FROM employee ORDER BY DEPT;
```

On successful execution of the query, you get to see the following response:

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
+------+--------------+-------------+-------------------+--------+
```

JDBC Program

Here is the JDBC program to apply Order By clause for the given example.

```
import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;



public class HiveQLOrderBy {
```

```
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";


   public static void main(String[] args) throws SQLException {


      // Register driver and create driver instance

      Class.forName(driverName);


      // get connection

      Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "",
"");


      // create statement

      Statement stmt = con.createStatement();


      // execute statement

      Resultset res = stmt.executeQuery("SELECT * FROM employee ORDER BY DEPT;");

      System.out.println(" ID \t Name \t Salary \t Designation \t Dept ");


      while (res.next()) {

         System.out.println(res.getInt(1) + " " + res.getString(2) + " " + res.getDouble(3) + " " +
res.getString(4) + " " + res.getString(5));

      }


      con.close();

   }

}
```

Save the program in a file named HiveQLOrderBy.java. Use the following commands to compile and execute this program.

```
$ javac HiveQLOrderBy.java

$ java HiveQLOrderBy
```

Output:

```
ID       Name           Salary      Designation          Dept
1205     Kranthi        30000       Op Admin             Admin
1204     Krian          40000       Hr Admin             HR
1202     Manisha        45000       Proofreader          PR
1201     Gopal          45000       Technical manager    TP
1203     Masthanvali    40000       Technical writer     TP
1204     Krian          40000       Hr Admin             HR
```

HiveQL - Select-Group By

The GROUP BY clause is used to group all the records in a result set using a particular collection column. It is used to query a group of records.

Syntax

The syntax of GROUP BY clause is as follows:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference

[WHERE where_condition]

[GROUP BY col_list]

[HAVING having_condition]

[ORDER BY col_list]]

[LIMIT number];
```

Example

The following query retrieves the employee details using the above scenario.

```
hive> SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

On successful execution of the query, you get to see the following response:

```
+------+--------------+
| Dept | Count(*)     |
+------+--------------+
|Admin |    1         |
|PR    |    2         |
|TP    |    3         |
+------+--------------+
```

JDBC Program

Given below is the JDBC program to apply the Group By clause for the given example.

```
import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;



public class HiveQLGroupBy {

   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
```

```
   public static void main(String[] args) throws SQLException {

      // Register driver and create driver instance
      Class.forName(driverName);

      // get connection
      Connection con = DriverManager.
      getConnection("jdbc:hive://localhost:10000/userdb", "", "");

      // create statement
      Statement stmt = con.createStatement();

      // execute statement
      Resultset res = stmt.executeQuery("SELECT Dept,count(*) " + "FROM employee GROUP BY DEPT; ");
      System.out.println(" Dept \t count(*)");

      while (res.next()) {
         System.out.println(res.getString(1) + " " + res.getInt(2));
      }
      con.close();
   }
}
```

Save the program in a file named HiveQLGroupBy.java. Use the following commands to compile and execute this program.

```
$ javac HiveQLGroupBy.java

$ java HiveQLGroupBy
```

Output:

```
 Dept     Count(*)
 Admin       1
 PR          2
 TP          3
```

HiveQL - Select-Joins

JOIN is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from

two or more tables in the database. It is more or less similar to SQL JOIN.

Syntax

```
join_table:


   table_reference JOIN table_factor [join_condition]

   | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference

   join_condition

   | table_reference LEFT SEMI JOIN table_reference join_condition

   | table_reference CROSS JOIN table_reference [join_condition]
```

Example

Consider the following table named CUSTOMERS..

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

Consider another table ORDERS as follows:

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 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   |
+-----+---------------------+-------------+--------+
```

There are different types of joins given as follows:

- JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

JOIN

JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables.

The following query executes JOIN on the CUSTOMER and ORDER tables, and retrieves the records:

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

On successful execution of the query, you get to see the following response:

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | Khilan   | 25  | 1560   |
| 4  | Chaitali | 25  | 2060   |
+----+----------+-----+--------+
```

LEFT OUTER JOIN

The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table.

A LEFT 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.

The following query demonstrates LEFT OUTER JOIN between CUSTOMER and ORDER tables:

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

On successful execution of the query, you get to see the following response:

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
| 1  | Ramesh   | NULL   | NULL                |
| 2  | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | Hardik   | NULL   | NULL                |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | NULL   | NULL                |
+----+----------+--------+---------------------+
```

RIGHT OUTER JOIN

The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

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.

The following query demonstrates RIGHT OUTER JOIN between the CUSTOMER and ORDER tables.

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

On successful execution of the query, you get to see the following response:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

FULL OUTER JOIN

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

The following query demonstrates FULL OUTER JOIN between CUSTOMER and ORDER tables:

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

On successful execution of the query, you get to see the following response:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 1    | Ramesh   | NULL   | NULL                |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5    | Hardik   | NULL   | NULL                |
| 6    | Komal    | NULL   | NULL                |
| 7    | Muffy    | NULL   | NULL                |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```