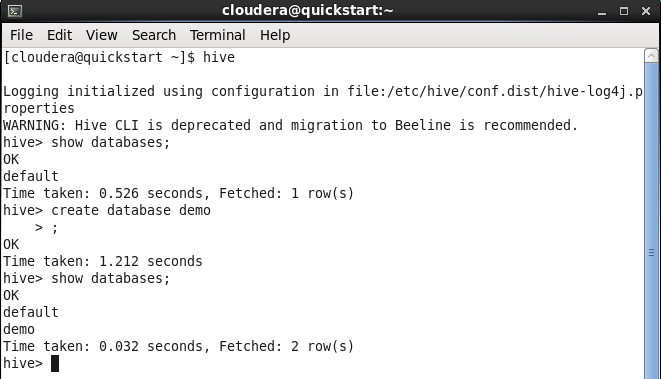
**Practical IV Hive commands**

// show databases:- To check default database provided by Hive

* **show databases;**

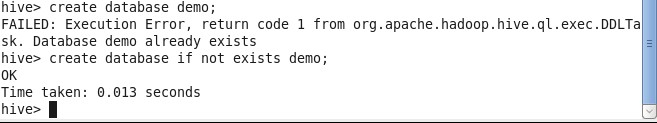
// create database:- To create a new database

* **create database [dbname];**



// if not exists:- If we want to suppress the warning generated by Hive on creating the database with the same name.

* **create database if not exists [dbname];**



// with dbproperties:- It allows assigning properties with the database in the form of key-value pair

* **create database [dbname] with dbproperties**

**(**

**‘creator’=’Max’, ‘date’=’1996-05-25’**

**);**

// describe: Used to retrieve the information associated with the database

* **describe database extended [dbname]**



// create an internal table:- To create table in the database

* **create table [dbname].[tablename] (**

**[column\_names ]**

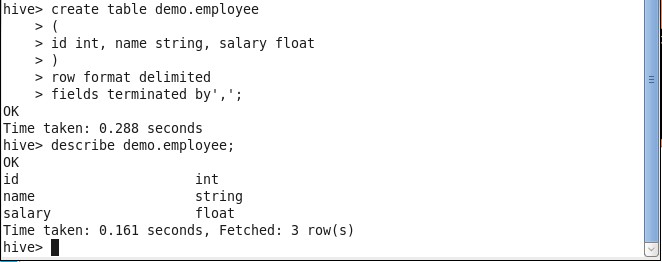
**);**

**row format delimited fields terminated by ‘,’;**

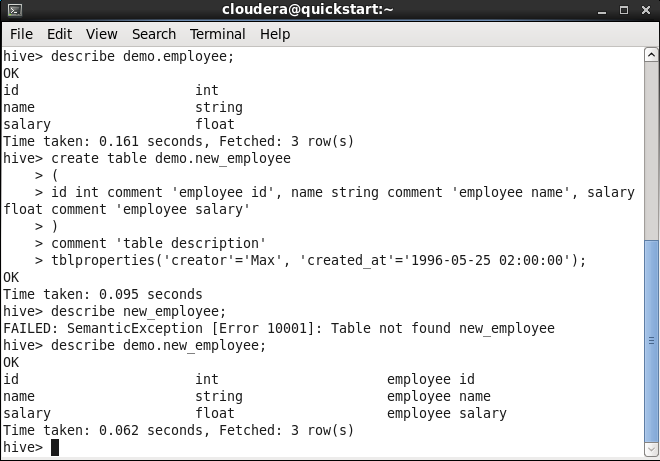
// describe table:- To see metadata or structure of the table

**Create table employee with fields: empid int, name string, salary float, deptid int**

* **describe [dbname].[tablename];**

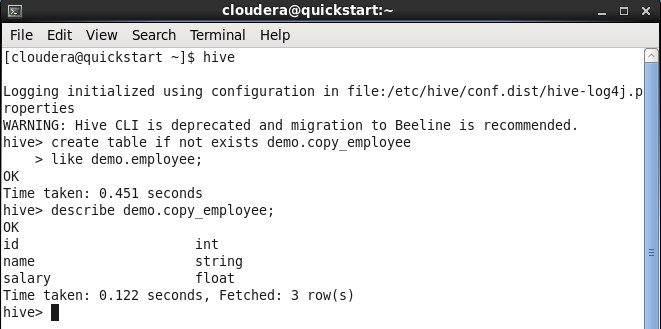


// comments:- while creating a table, we can add the comments to the columns and can also define the table properties



// create table if not exists:- Used to create a new table using the schema of an existing table

* **create table if not exists [dbname].[table\_name] like [dbname].[table\_name];**



Load Data:-

// loading file from local file system

**load data local inpath ‘e.g. any csv or txt file’ into table employee;**

// loading file from hdfs

**load data inpath ‘e.g. any csv or txt file’ into table employee;**

And check If it is loaded or not.

Command to exit from hive – **exit;**

**cat > employee.txt**

**1,cyrus,35000,1**

**2,keegun,50000,1**

**3,allen,40000,2**

Enter data in input file and press enter and ctrl z

Command: **cat employee.txt**

Move the **employee.txt** file to the Hadoop file system

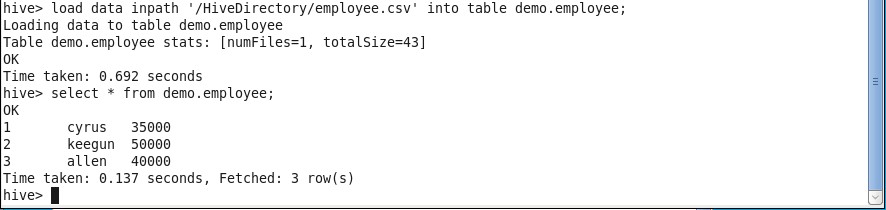
**hdfs dfs -put employee.txt /user/cloudera**

**hdfs dfs -ls**

Command to enter hive

**hive**

**load data inpath ‘employee.txt ’ into table demo.employee;**

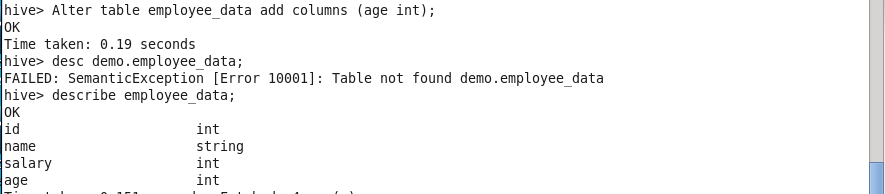


Alter Table:-

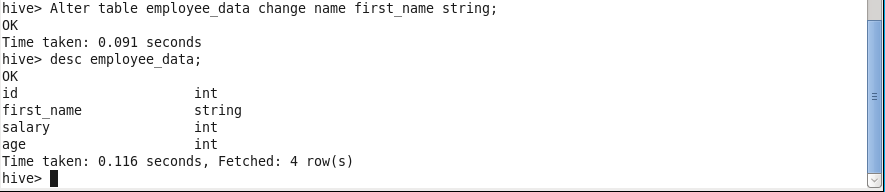
// Rename the table\_name



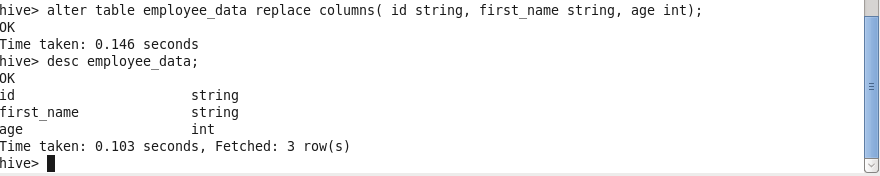
// add columns



// alter column\_name



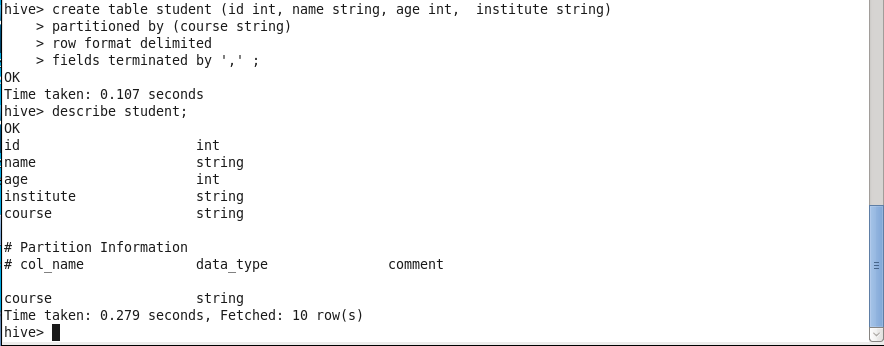
// delete/replace column



Static Partitioning:-

// Use the database in which you want to create a table

**create table student(id int, name string, age int, institute string) partitioned by(course string) row format delimited fields terminated by ‘,’;**



//creating files to store students’ data

**exit;**

**cat>student1.txt**

**1,keegun,25,khalsa,Hadoop**

**2,cyrus,24,vidyalankar,Hadoop**

**3,allen,26,ruia,Hadoop**

**cat>student2.txt**

**1,saurav,23,met,java**

**2,seta,25,met,java**

**3,geeta,22,vjti,java**

Move the **student1.txt** and **student2.txt** files to the Hadoop file system

**hdfs dfs -put student1.txt /user/cloudera**

**hdfs dfs –ls**

**hdfs dfs -put student2.txt /user/cloudera**

**hdfs dfs -ls**

Command to enter hive

**hive**

// Load data with partitioning and check the result by partition course field.

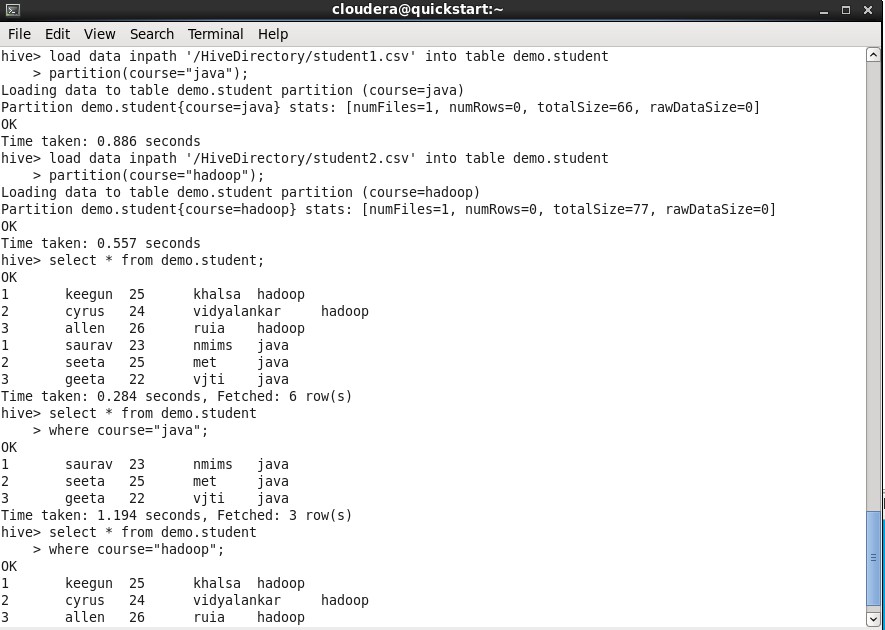
**load data inpath ‘student1.txt ’ into table demo.student partition(course=”java”);**

**load data inpath ‘student2.txt ’ into table demo.student partition(course=”hadoop”);**

**select\*from demo.student;**

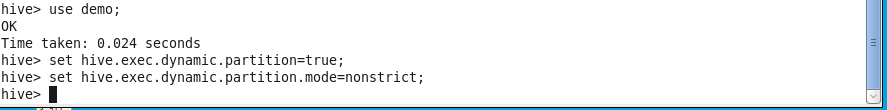
**select\*from demo.student where course=”java”;**

**select\*from demo.student where course=”hadoop”;**



Dynamic Partitioning:-

// First use the database you want and set the dynamic partition values as following:-



// Create one dummy table and load data into it

Hive;

Use demo;

**Set hive.exec.dynamic.partition=true;**

**Set hive.exec.dynamic.partition.mode=nonstrict;**

**exit;**

**cat>student\_details.txt**

**1,victoria,23,kms,hadoop**

**2,elizabeth,24,wlmt,java**

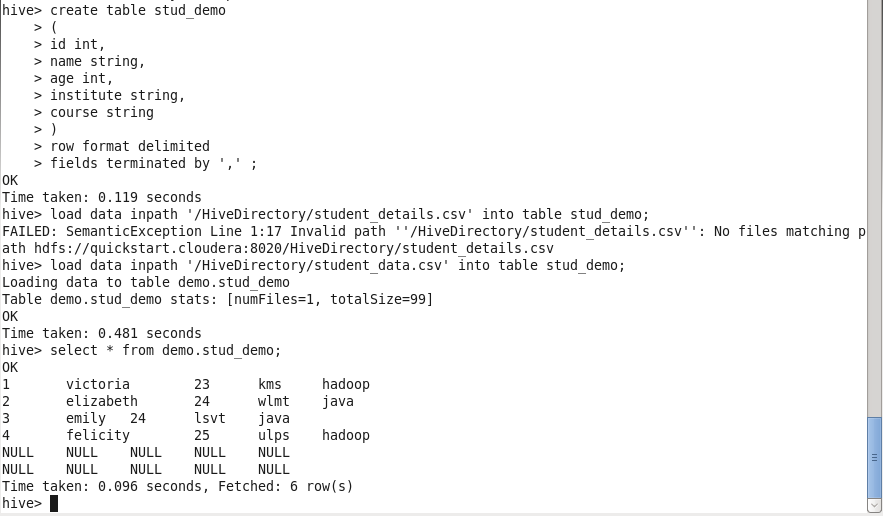
**3,emily,24,isvt,java**

**4,felicity,25,ulps,hadoop**

**Create table stud\_demo(id int, name string, age int, institute string, course string) row format delimited fields terminated by ‘,’;**

**Load data local inpath ‘/home/cloudera/student\_details.txt’ into table stud\_demo;**

**Select \*from stud\_demo;**



// Now create one partition table and insert data of dummy table stud\_demo into it.

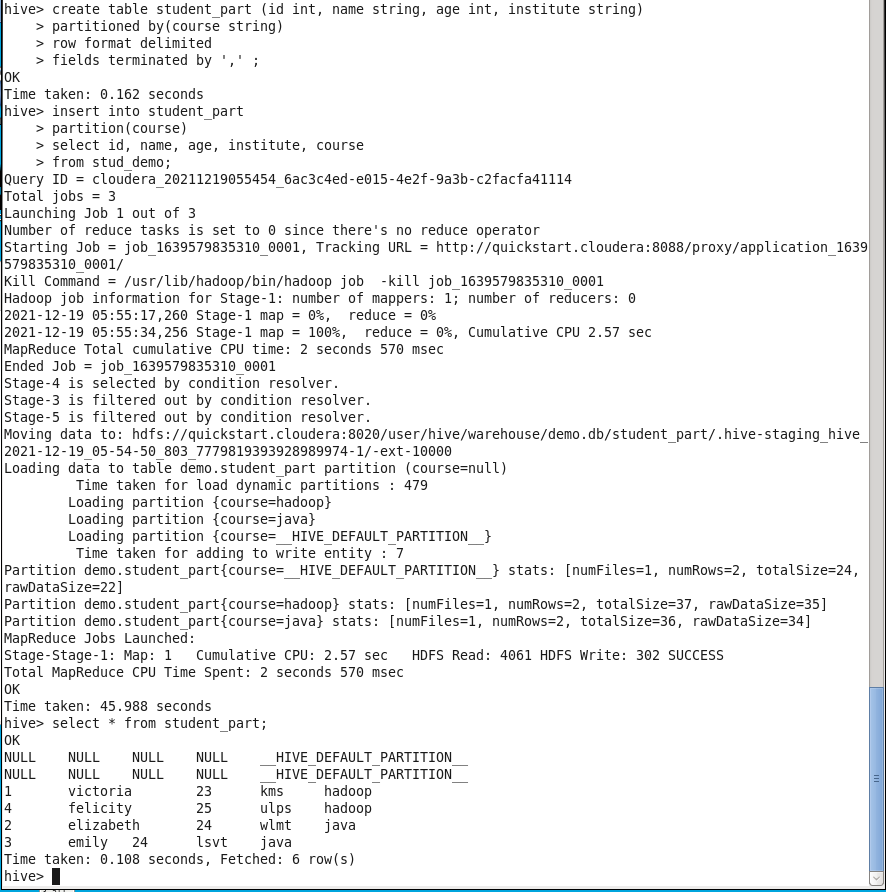
Create table student\_part(id int, name string, age int, institute string) partitioned by(course string)row format delimited fields terminated by ‘,’;

Insert into student\_part partition(course) select id, name, age, institute, course from stud\_demo;

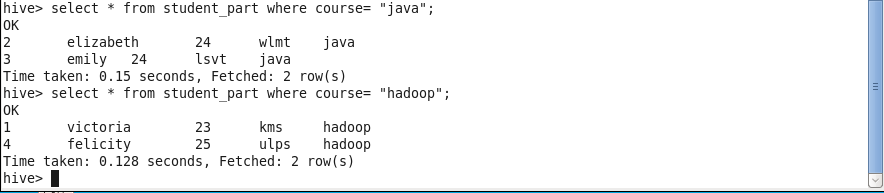
Select \* from student\_part;

Select \* from student\_part where course=”java”;

Select \* from student\_part where course=”hadoop”;



// Retrive the data by partition



HiveQL Operators:-

The HiveQL operators facilitate to perform various arithmetic and relational operations.

// Select the database

**use demo;**

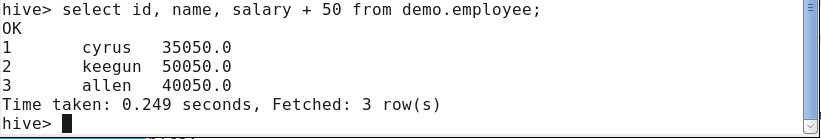
**desc employee;**

**select \*from employee;**

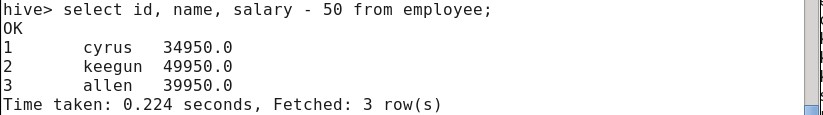
1. Arithmetic Operators:-

In Hive, the arithmetic operator accepts any numeric type

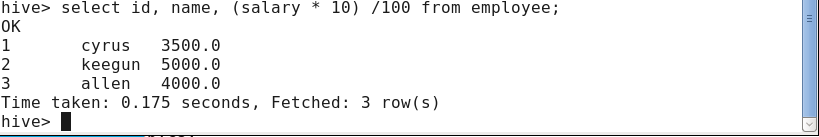
// Let's see an example to increase the salary of each employee by 50



// Let's see an example to decrease the salary of each employee by 50



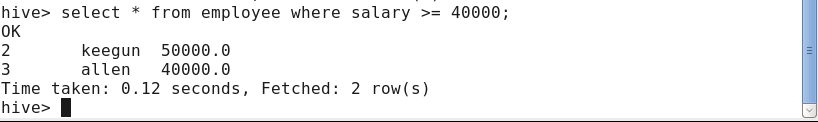
// Let's see an example to find out the 10% salary of each employee



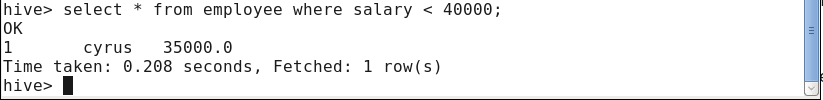
1. Relational Operators:-

In Hive, the relational operators are generally used with clauses like Join and Having to compare the existing records.

// Let's see an example to fetch the details of the employee having salary>=40000



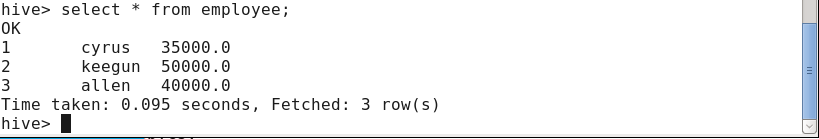
// Let's see an example to fetch the details of the employee having salary<40000



HiveQL Functions:-

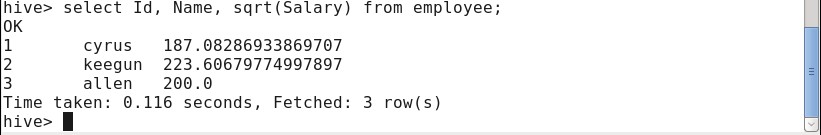
The Hive provides various in-built functions to perform mathematical and aggregate type operations.

// Select the database in which we want to create a table, load the data into the table and fetch the loaded data by using the following command:



1. Mathematical Functions:

// Let's see an example to fetch the square root of each employee's salary.

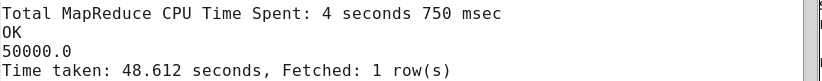


1. Aggregate Functions:

In Hive, the aggregate function returns a single value resulting from computation over many rows.

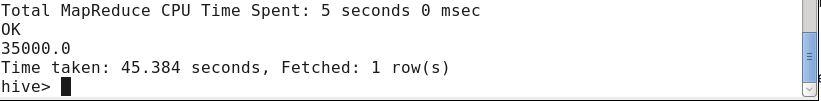
// Let's see an example to fetch the maximum salary of an employee.

select max(salary) from employee;



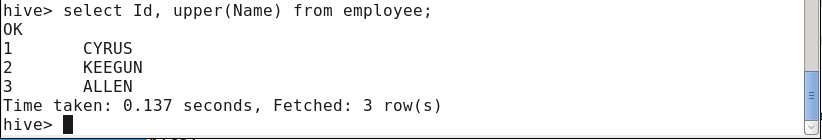
// Let's see an example to fetch the minimum salary of an employee

select min(salary) from employee;

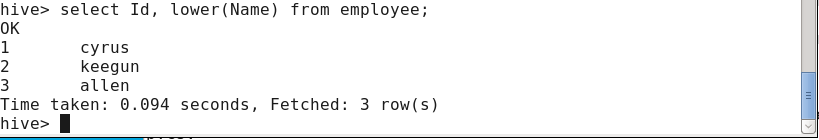


1. Built-in Functions:-

// Let's see an example to fetch the name of each employee in uppercase



// Let's see an example to fetch the name of each employee in lowercase



HiveQL Group By and Having Clause

The Hive Query Language provides GROUP BY and HAVING clauses that facilitate similar functionalities as in SQL.

1. Group By Clause:

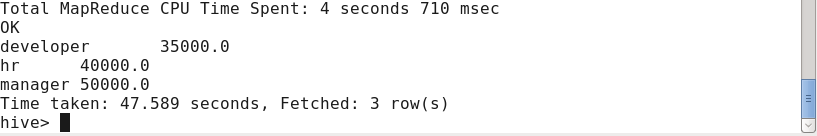
The HQL Group By clause is used to group the data from the multiple records based on one or more column. It is generally used in conjunction with the aggregate functions (like SUM, COUNT, MIN, MAX and AVG) to perform an aggregation over each group

// Select the database in which we want to create a table, load the data into the table and fetch the loaded data by using the following command:



// Now, fetch the sum of employee salaries department wise by using the following command

**select department,sum(salary) from emp group by department**

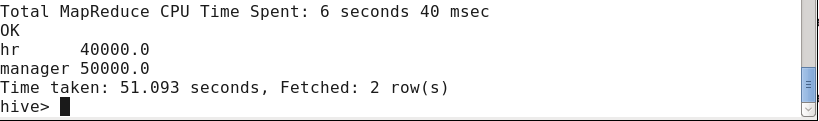


1. Having Clause:

The HQL HAVING clause is used with GROUP BY clause. Its purpose is to apply constraints on the group of data produced by GROUP BY clause. Thus, it always returns the data where the condition is TRUE

// In this example, we fetch the sum of employee's salary based on department and apply the required constraints on that sum by using HAVING clause. Let's fetch the sum of employee's salary based on department having sum >= 40000 by using the following command

**select department,sum(salary) from emp group by department having sum(salary)>=40000**



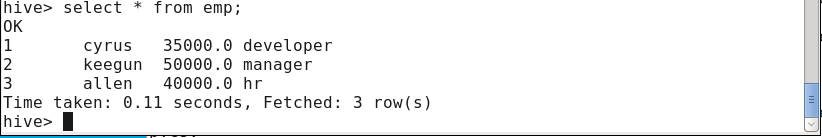
HiveQL Order By and Sort By Clause:-

By using HiveQL ORDER BY and SORT BY clause, we can apply sort on the column. It returns the result set either in ascending or descending order.

1. Order By Clause:

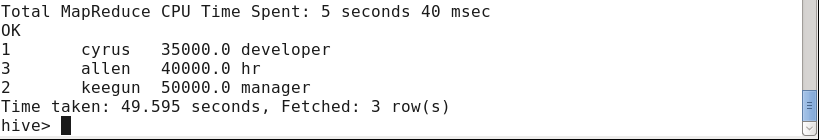
// In HiveQL, ORDER BY clause performs a complete ordering of the query result set. Hence, the complete data is passed through a single reducer. This may take much time in the execution of large datasets. However, we can use LIMIT to minimize the sorting time. Let's see an example to arrange the data in the sorted order by using ORDER BY clause.

// Select the database in which we want to create a table, load the data into the table and fetch the loaded data by using the following command:



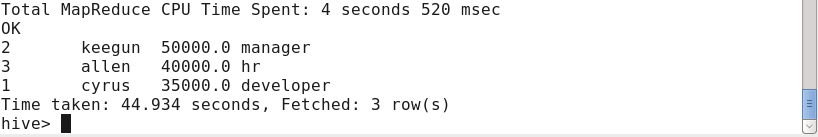
// Let's see an example to arrange the data in the ascending order by using ORDER BY clause

**select \*from emp order by salary**



// Now, fetch the data in the descending order by using the following command

**select \* from emp order by salary desc**

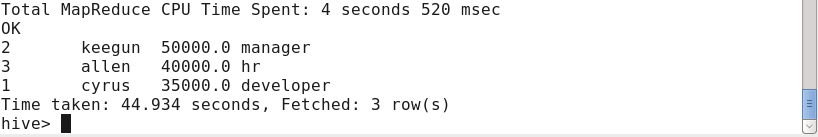


1. Sort By Clause:

The HiveQL SORT BY clause is an alternative of ORDER BY clause. It orders the data within each reducer. Hence, it performs the local ordering, where each reducer's output is sorted separately. It may also give a partially ordered result.

// Let's fetch the data in the descending order by using the following command

**select \*from emp sort by salary desc**



HiveQL Join:-

create table employee1(empid int, empname string, deptid int) row format delimited fields terminated by ‘,’;

create table dept(deptid int, department\_name string) row format delimited fields terminated by ‘,’;

load data local inpath ‘/home/cloudera/join1.txt’ into table employee1;

load data local inpath ‘/home/cloudera/join2.txt’ into table dept;

select \*from employee1;

1 akhila 2

2 preeti 2

3 ria 1

4 alexa 4

5 meena 3

6 heena 6

select \* from dept;

1 it

2 sales

3 hr

4 marketing

5 manufacturin

**set hive.auto.convert.join=false**

Inner join

select e1.empname, e2.department\_name from employee1 e1 join dept e2 on e1.deptid=e2.deptid;

Left outer join

select e1.empname, e2.department\_name from employee1 e1 left outer join dept e2 on e1.deptid=e2.deptid;

Right outer join

select e1.empname, e2.department\_name from employee1 e1 right outer join dept e2 on e1.deptid=e2.deptid;

Full outer join

select e1.empname, e2.department\_name from employee1 e1 full outer join dept e2 on e1.deptid=e2.deptid

**view**

Q.Create view on employee table for employee whose salary greater than 2000 and then drop that view.

>create view sal as select \* from stud\_demo/employee where age/salary>40000.0;

>select \* from sal;

>drop view sal;

Q. Create index on employee table then drop that index.

**index**

>create index empindex1 on table employee(empname) as

‘org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler’ with deferred rebuild;

>show index on employee;

>drop index empindex1 on employee;

>show index on employee;