**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.

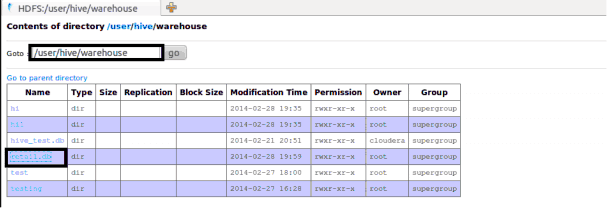
Go to Hive shell by giving the command sudo hive and enter the command**‘create database<data base** **name>’** to create the new database in the Hive.



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

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/21-1.png)

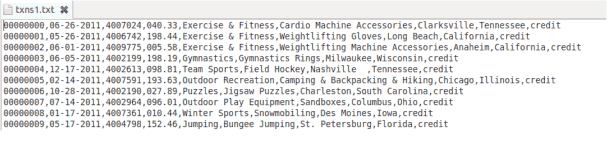
The database creates in a default location of the Hive warehouse. In Cloudera, Hive database store in a /user/hive/warehouse.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/22-1.png)

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

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/23-1.png)

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

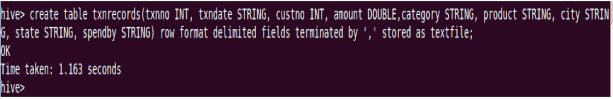
[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/24-1.png)

Hive table

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.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/26-1.png)

**Describe** provides information about the schema of the table.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/27-1.png)

1. Hive Data Manipulations

Solutions:

**Data Manipulation Language (DML )**

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

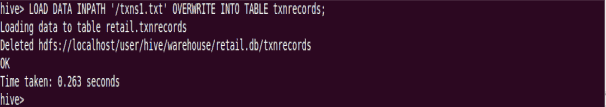
Example :

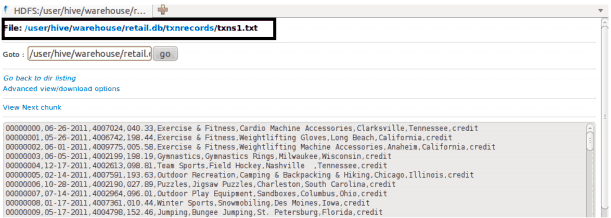
LOAD, INSERT Statements.

Syntax :

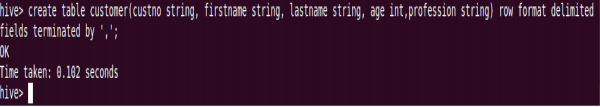
LOAD data <LOCAL> inpath <file path> into table [tablename]

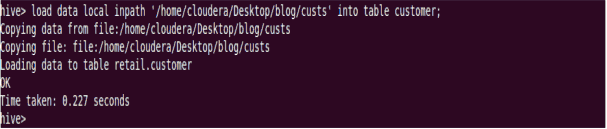
The Load operation is used to move the data into corresponding Hive table. If the keyword **local** is specified, then in the load command will give the local file system path. If the keyword local is not specified we have to use the HDFS path of the file.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/28-1.png)

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/29-1.png)

Here are some examples for the LOAD data LOCAL command

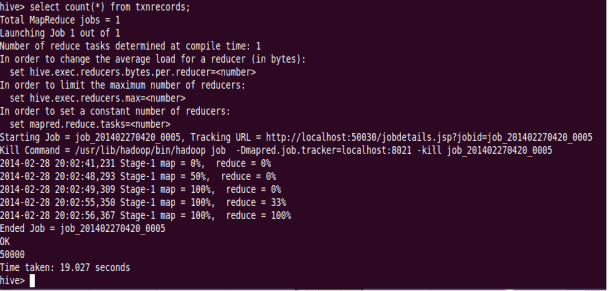
[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/30-1.png)

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/311-1.png)

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

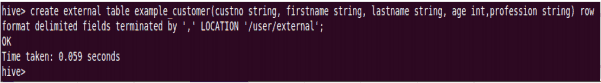
*Example to count number of records:*

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

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/32-1.png)

**‘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.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/33-1.png)

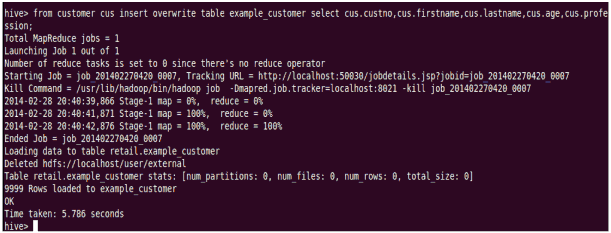
[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/34-1.png)

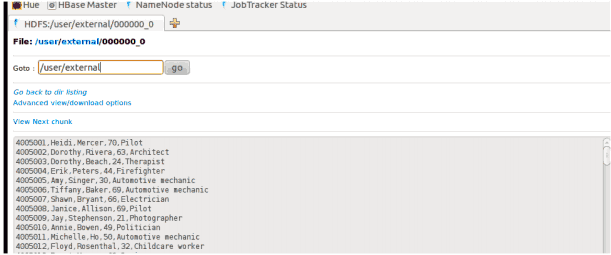
**Insert Command:**

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.

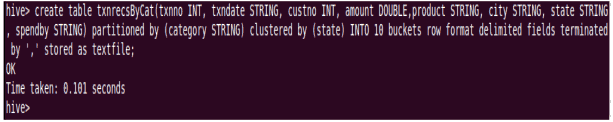
• INSERT INTO is used to append the data into existing data in a table. (Note: INSERT INTO syntax is work from the version 0.8)

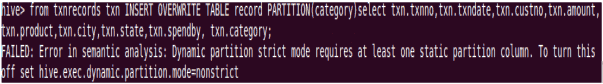
[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/35-1.png)

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/36-1.png)

**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.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/37-1.png)

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/38-1.png)

When we insert the data Hive throwing errors, the dynamic partition mode is strict and dynamic partition not enabled (by [Jeff](http://www.dresshead.com/dresshead-staff-profile-jeff-maurer/) at [dresshead website](http://www.dresshead.com/)). So we need to set the following parameters in Hive shell.

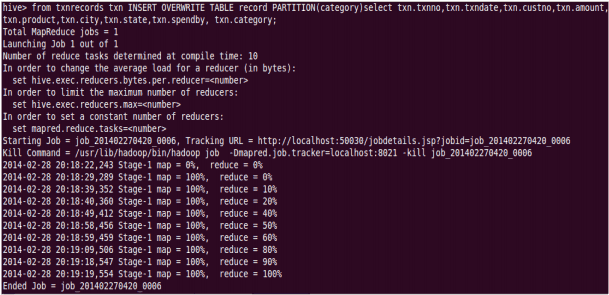
set hive.exec.dynamic.partition=true;

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

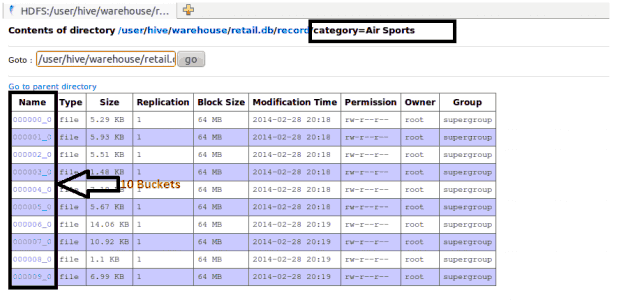
set hive.exec.dynamic.partition.mode=nonstrict;

[Dynamic Partitions](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/39-1.png)

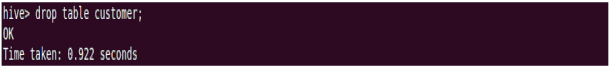
[Dynamic Partitions](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/40-1.png)

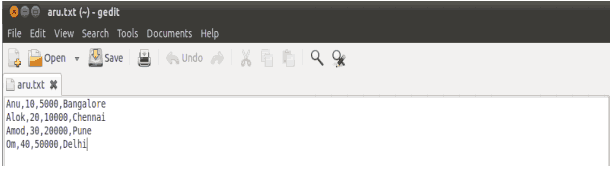
[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/41-1.png)

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

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/42-1.png)

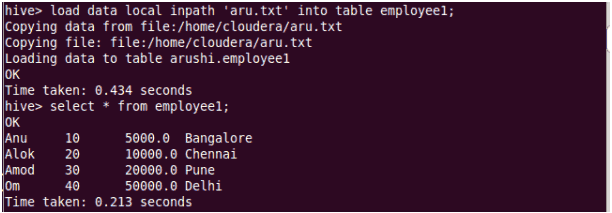
The ‘Drop Table’ statement deletes the data and metadata for a table. In the case of external tables, only the metadata is deleted.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/43-1.png)

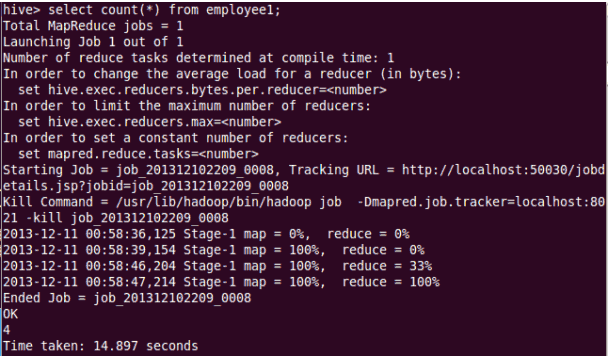
[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/44-1.png)

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

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/45-1.png)

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

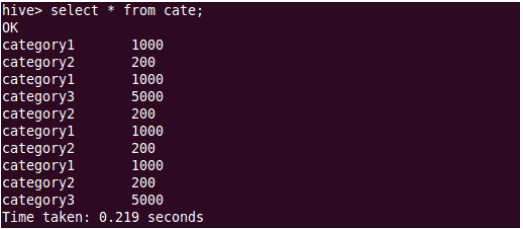
[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/46-1.png)

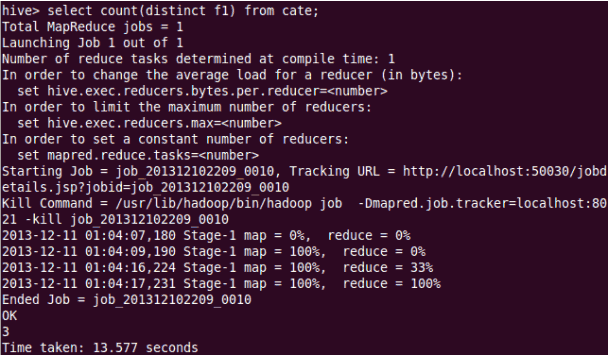
**Aggregation :**

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.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/47-1.png)

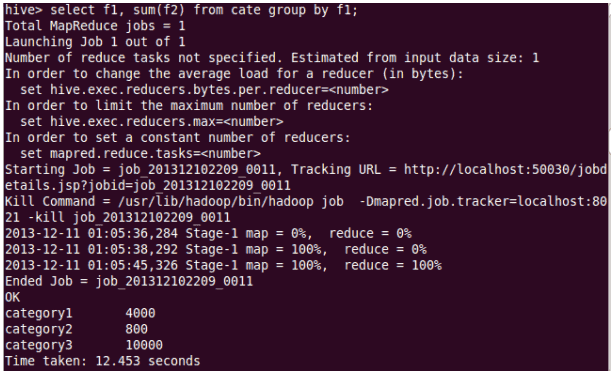
[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/48-1.png)

**Grouping :**

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

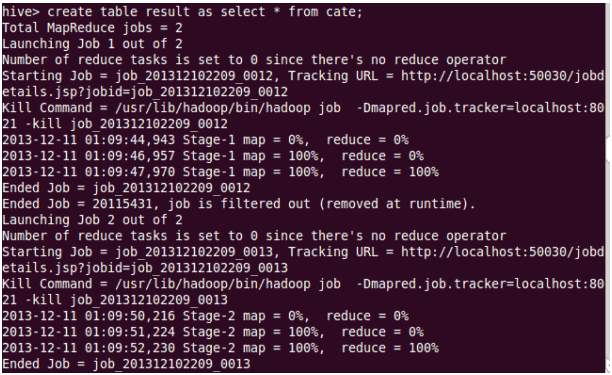
Select category, sum( amount) from txt records group by category

It calculates the amount of same category.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/49-11.png)

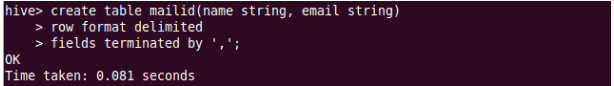
The result one table is stored in to another table.

Create table newtablename as select \* from oldtablename;

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/50-1.png)

**Join Command :**

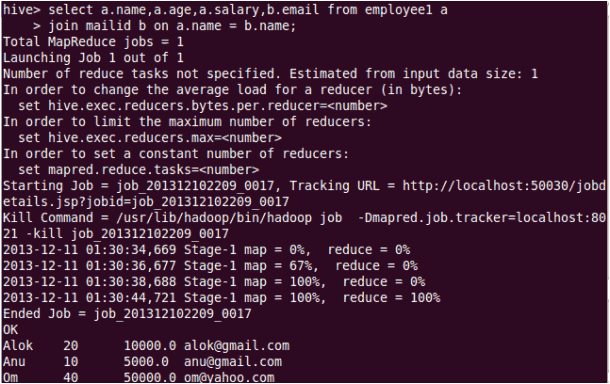
Here one more table is created in the name**‘mailid’**

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/51-1.png)

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/52-1.png)

**Join Operation**:

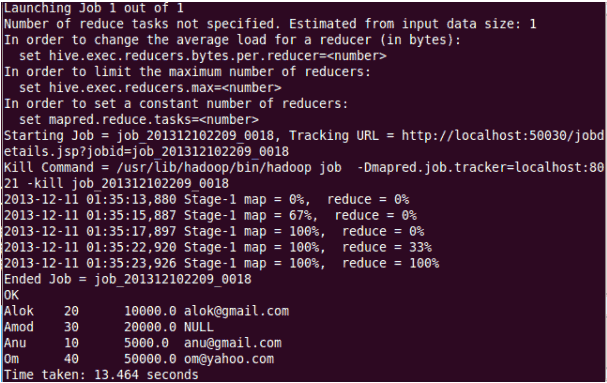
A Join operation is performed to combining fields from two tables by using values common to each.

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/53-1.png)

**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).

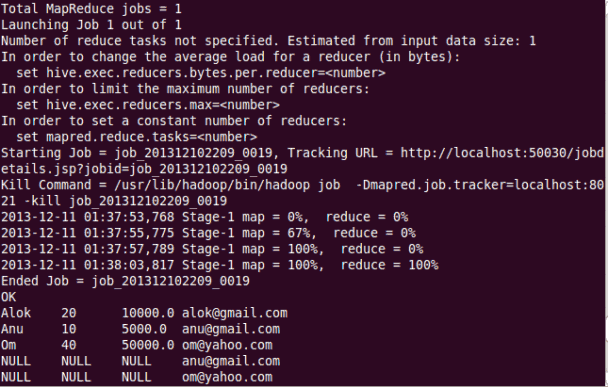
[Left Outer Join](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/54-1.png)

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/55-1.png)

**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.

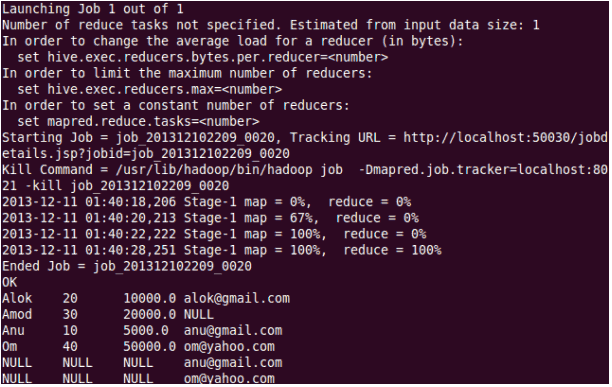
[Right Outer Join](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/56-1.png)

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/57-1.png)

**Full Join**:

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

[Full Join](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/58-1.png)

[](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/59-1.png)

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

[Exiting from Hive](https://cdn.edureka.co/blog/wp-content/uploads/2014/03/60-1.png)

1. 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 |

+------+----------+--------+---------------------+