**HiveQL [Hive Query Language]**

1. **HiveQL Data Definition**

Hive Data Definition language is used for creating, altering, and dropping databases, tables, views, functions, and indexes.

HiveQL DDL statements are as follows:

* **CREATE** DATABASE/SCHEMA, TABLE, VIEW, FUNCTION, INDEX
* **DROP** DATABASE/SCHEMA, TABLE, VIEW, INDEX
* **TRUNCATE** TABLE
* **ALTER** DATABASE/SCHEMA, TABLE, VIEW
* **MSCK REPAIR** TABLE (or ALTER TABLE RECOVER PARTITIONS)
* **SHOW** DATABASES/SCHEMAS, TABLES, TBLPROPERTIES, VIEWS, PARTITIONS, FUNCTIONS, INDEX[ES], COLUMNS, CREATE TABLE
* **DESCRIBE** DATABASE/SCHEMA, table\_name, view\_name

PARTITION statements are usually options of TABLE statements, except for SHOW PARTITIONS.

## Create/Describe/Show/Drop/Alter/Use Database

**Create Database**

**Syntax:**

CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database\_name

  [COMMENT database\_comment]

  [LOCATION hdfs\_path]

  [WITH DBPROPERTIES (property\_name=property\_value, ...)];

**Example:**

**hive> CREATE DATABASE acadgild;**

Hive will throw an error if “acadgild” table already exists. We can suppress these warnings with following command:

**hive> CREATE DATABASE IF NOT EXISTS acadgild;**

One can use the keyword SCHEMA instead of DATABASE in all the database-related commands.

**Describe Database**

**Syntax**

DESCRIBE DATABASE [EXTENDED] db\_name;

DESCRIBE DATABASE shows the name of the database, its comment (if one has been set), and its root location on the filesystem. EXTENDED also shows the [database properties](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateDatabase).

**Example**

**hive> Descibe acadgild;**

**Show Databases**

**Syntax:**

**SHOW DATABASES [LIKE ‘identifier\_with\_wildcards’];**

SHOW DATABASES lists all of the databases defined in the metastore. The optional LIKE clause allows the list of databases to be filtered using a regular expression. Wildcards in the regular expression can only be '\*' for any character(s) or '|' for a choice. Examples are 'employees', 'emp\*', 'emp\*|\*ees', all of which will match the database named 'employees'.

**Example:**

**hive> SHOW DATABASES;**

default

acadgild

**Use Database**

**Syntax**

**USE database\_name;**

USE sets the current database for all subsequent HiveQL statements. To revert to the default database, use the keyword "default" instead of a database name. To check which database is currently being used: SELECT [current database()](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Misc.Functions) (as of Hive 0.13.0).

**Example**

**hive> USE acadgild;**

**Alter Database**

**Syntax**

ALTER DATABASE database\_name SET DBPROPERTIES (property\_name=property\_value, ...);

 ALTER DATABASE database\_name SET OWNER [USER|ROLE] user\_or\_role;

ALTER DATABASE database\_name SET LOCATION hdfs\_path;

The ALTER DATABASE ... SET LOCATION statement does not move the contents of the database's current directory to the newly specified location. It does not change the locations associated with any tables/partitions under the specified database. It only changes the default parent-directory where new tables will be added for this database. This behaviour is analogous to how changing a table-directory does not move existing partitions to a different location.

No other metadata about a database can be changed.

**Example**

**hive> Alter Database acadgild SET DBPROPERTIES (‘Modified by’ = ‘Deepika’);**

**Drop Database:**

**Syntax**

**DROP DATABASE [IF EXISTS] database\_name [RESTRICT | CASCADE];**

DROP Database command is used to drop the database and its metadata. The default behaviour is RESTRICT, where DROP DATABASE will fail if the database is not empty. To drop the tables in the database as well, use DROP DATABASE ... CASCADE.

**Example**

**hive> Drop database acadgild;**

## Create/Describe/Show/Alter/Drop/Truncate Table

**Create Table**

**Syntax**

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db\_name.]table\_name    -- (Note: TEMPORARY available in Hive 0.14.0 and later)

  [(col\_name data\_type [COMMENT col\_comment], ... [constraint\_specification])]

  [COMMENT table\_comment]

  [PARTITIONED BY (col\_name data\_type [COMMENT col\_comment], ...)]

  [CLUSTERED BY (col\_name, col\_name, ...) [SORTED BY (col\_name [ASC|DESC], ...)] INTO num\_buckets BUCKETS]

  [SKEWED BY (col\_name, col\_name, ...)                  -- (Note: Available in Hive 0.10.0 and later)]

     ON ((col\_value, col\_value, ...), (col\_value, col\_value, ...), ...)

     [STORED AS DIRECTORIES]

  [

   [ROW FORMAT row\_format]

   [STORED AS file\_format]

     | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]  -- (Note: Available in Hive 0.6.0 and later)

  ]

  [LOCATION hdfs\_path]

  [TBLPROPERTIES (property\_name=property\_value, ...)]   -- (Note: Available in Hive 0.6.0 and later)

  [AS select\_statement];   -- (Note: Available in Hive 0.5.0 and later; not supported for external tables)

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db\_name.]table\_name

  LIKE existing\_table\_or\_view\_name

  [LOCATION hdfs\_path];

data\_type

  : primitive\_type

  | array\_type

  | map\_type

  | struct\_type

  | union\_type  -- (Note: Available in Hive 0.7.0 and later)

primitive\_type

  : TINYINT

  | SMALLINT

  | INT

  | BIGINT

  | BOOLEAN

  | FLOAT

  | DOUBLE

  | DOUBLE PRECISION -- (Note: Available in Hive 2.2.0 and later)

  | STRING

  | BINARY      -- (Note: Available in Hive 0.8.0 and later)

  | TIMESTAMP   -- (Note: Available in Hive 0.8.0 and later)

  | DECIMAL     -- (Note: Available in Hive 0.11.0 and later)

  | DECIMAL(precision, scale)  -- (Note: Available in Hive 0.13.0 and later)

  | DATE        -- (Note: Available in Hive 0.12.0 and later)

  | VARCHAR     -- (Note: Available in Hive 0.12.0 and later)

  | CHAR        -- (Note: Available in Hive 0.13.0 and later)

array\_type

  : ARRAY < data\_type >

map\_type

  : MAP < primitive\_type, data\_type >

struct\_type

  : STRUCT < col\_name : data\_type [COMMENT col\_comment], ...>

union\_type

   : UNIONTYPE < data\_type, data\_type, ... >  -- (Note: Available in Hive 0.7.0 and later)

row\_format

  : DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]] [COLLECTION ITEMS TERMINATED BY char]

        [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]

        [NULL DEFINED AS char]   -- (Note: Available in Hive 0.13 and later)

  | SERDE serde\_name [WITH SERDEPROPERTIES (property\_name=property\_value, property\_name=property\_value, ...)]

file\_format:

  : SEQUENCEFILE

  | TEXTFILE    -- (Default, depending on hive.default.fileformat configuration)

  | RCFILE      -- (Note: Available in Hive 0.6.0 and later)

  | ORC         -- (Note: Available in Hive 0.11.0 and later)

  | PARQUET     -- (Note: Available in Hive 0.13.0 and later)

  | AVRO        -- (Note: Available in Hive 0.14.0 and later)

  | INPUTFORMAT input\_format\_classname OUTPUTFORMAT output\_format\_classname

constraint\_specification:

  : [, PRIMARY KEY (col\_name, ...) DISABLE NOVALIDATE ]

    [, CONSTRAINT constraint\_name FOREIGN KEY (col\_name, ...) REFERENCES table\_name(col\_name, ...) DISABLE NOVALIDATE

CREATE TABLE creates a table with the given name. An error is thrown if a table or view with the same name already exists. We can use IF NOT EXISTS to skip the error.

**Example**  
  
CREATE TABLE IF NOT EXISTS employees (  
name  STRING COMMENT 'Employee name',  
salary FLOAT COMMENT 'Employee salary',  
subordinates ARRAY COMMENT 'Names of subordinates',  
deductions MAP COMMENT 'Keys are deductions names, values are percentages',  
address STRUCT  
COMMENT 'Home address')  
COMMENT 'Employees Table'  
TBLPROPERTIES ('creator'='Deepika', 'created\_at'='2017-08-18 10:00:00’)  
LOCATION '/user/hive/warehouse/emp.db/employees';

#### Create Table As Select (CTAS)

Tables can also be created and populated by the results of a query in one create-table-as-select (CTAS) statement. The table created by CTAS is atomic, meaning that the table is not seen by other users until all the query results are populated. So other users will either see the table with the complete results of the query or will not see the table at all.

There are two parts in CTAS, the SELECT part can be any SELECT statement supported by HiveQL. The CREATE part of the CTAS takes the resulting schema from the SELECT part and creates the target table with other table properties such as the SerDe and storage format.

CTAS has these restrictions:

* The target table cannot be a partitioned table.
* The target table cannot be an external table.
* The target table cannot be a list bucketing table.

**Example**

CREATE TABLE acadgild\_big\_data\_new

   ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"

   STORED AS RCFile

   AS

SELECT (ID) studentID, concat(courseName, courseFee) courseNameFee

FROM acadgild\_big\_data

SORT BY studentID, courseNameFee;

The above CTAS statement creates the target table “acadgild\_big\_data\_new” with the schema (studentID INT, courseNameFee STRING) derived from the results of the SELECT statement. If the SELECT statement does not specify column aliases, the column names will be automatically assigned to \_col0, \_col1, and \_col2 etc. In addition, the new target table is created using a specific SerDe and a storage format independent of the source tables in the SELECT statement.

#### Create Table Like

The LIKE form of CREATE TABLE allows **to copy an existing table definition exactly (without copying its data).** In contrast to CTAS, the statement below creates a new acadgild\_big\_data\_new table whose definition exactly matches the existing acadgild\_big\_data in all particulars other than table name. The new table contains no rows.

**Example**

CREATE TABLE acadgild\_big\_data\_new

LIKE acadgild\_big\_data;

**Managed and External Tables**

**Managed/Internal Table**

* By default Hive creates managed tables, where files, metadata and statistics are managed by internal Hive processes.
* A managed table is stored under **hive.metastore.warehouse.dir** path property, by default in a folder path similar to /**apps/hive/warehouse/databasename.db/tablename/**.
* The default location can be overridden by the location property during table creation.
* If a managed table or partition is dropped, the data and metadata associated with that table or partition are deleted. If the PURGE option is not specified, the data is moved to a trash folder for a defined duration.
* Managed tables are used when Hive should manage the lifecycle of the table, or when generating temporary tables.

**Create Managed Table**

**Example**

hive> create table emp(id int, name string, salary float)

> row format delimited

> fields terminated by ‘\t’ ;

**NOTE: When we drop a managed table Hive deletes the data in the table**  
  
**External Table**

* An external table describes the metadata / schema on external files. External table files can be accessed and managed by processes outside of Hive.
* External tables can access data stored in sources such as Azure Storage Volumes (ASV) or remote HDFS locations.
* If the structure or partitioning of an external table is changed, an **MSCK REPAIR TABLE table\_name** statement can be used to refresh metadata information.
* External tables are used when files are already present or in remote locations, and the files should remain even if the table is dropped.

**Create External Table**

**Example**

hive> CREATE EXTERNAL TABLE IF NOT EXISTS stocks (  
> exchange STRING,  
> symbol STRING,  
> ymd STRING,  
> price\_open FLOAT,  
> price\_high FLOAT,  
> price\_low FLOAT,  
> price\_close FLOAT,  
> volume INT,  
> price\_adj\_close FLOAT)  
> ROW FORMAT DELIMITED  
> FIELDS TERMINATED BY ','  
> LOCATION '/data/stocks';

The EXTERNAL keyword lets us create a table and we need to provide a LOCATION so that Hive does not use a default location for this table.

When we drop an EXTERNAL table, data in the table is NOT deleted from the file system.

An EXTERNAL table points to any HDFS location for its storage, rather than being stored in a folder specified by the configuration property hive.metastore.warehouse.dir.

**NOTE:**

* Managed or external tables can be identified using the **DESCRIBE FORMATTED table\_name** command, which will display either **MANAGED\_TABLE** or **EXTERNAL\_TABLE** depending on table type.
* Statistics can be managed on internal and external tables and partitions for query optimization.

#### Partitioned Tables

* Partitioned tables can be created using the PARTITIONED BY clause.
* A table can have one or more partition columns and a separate data directory is created for each distinct value combination in the partition columns.
* Further, tables or partitions can be bucketed using CLUSTERED BY columns, and data can be sorted within that bucket via SORT BY columns. This can improve performance on certain kinds of queries.
* If, when creating a partitioned table, we get this error: "FAILED: Error in semantic analysis: Column repeated in partitioning columns," it means we are trying to include the partitioned column in the data of the table itself. We probably really do have the column defined. However, the partition we create makes a pseudocolumn on which we can query, so we must rename table column to something else (that users should not query on).

**Create Partitioned Table**

**Example**

hive> CREATE TABLE page\_view(viewTime INT, userid BIGINT,

 >  page\_url STRING, referrer\_url STRING,

 >  ip STRING COMMENT 'IP Address of the User')

 > COMMENT 'This is the page view table'

 > PARTITIONED BY(dt STRING, country STRING)

 > ROW FORMAT DELIMITED

 > FIELDS TERMINATED BY '\001'

> STORED AS SEQUENCEFILE;

The statement above creates the page\_view table with viewTime, userid, page\_url, referrer\_url, and ip columns (including comments). The table is also partitioned and data is stored in sequence files. The data format in the files is assumed to be field-delimited by ctrl-A and row-delimited by newline.

**Describe Table**

**Example**

hive> DESCRIBE empdb.employees;

or

hive> USE empdb;

hive> DESCRIBE employees;

**Example: Describe with Extended**

hive> DESCRIBE EXTENDED empdb.employees;  
name string Employee name  
salary float Employee salary  
subordinates array Names of subordinates  
deductions map Keys are deductions names, values are percentages  
address struct Home address  
Detailed Table Information Table(tableName:employees, dbName:empdb, owner:deepika,  
...  
location:hdfs://master-server/user/hive/warehouse/empdb.db/employees,  
parameters:{creator=Deepika, created\_at='2017-08-18 10:00:00',  
last\_modified\_user=Deepika, last\_modified\_time=1337544510,  
comment:Description of the table, ...}, ...)  
  
Note: Replacing **EXTENDED with FORMATTED** provides more readable but also more verbose output.  
  
**Show Table**

**Example**

hive> USE empdb;  
hive> SHOW TABLES;  
employees  
  
  
**Example: If we aren’t in the same database, we can still list the tables in that database**  
hive> USE default;  
hive> SHOW TABLES IN mydb;  
employees  
table1  
table2  
  
**Example: With lot of tables, we can limit the ones listed using a regular expression**  
hive> USE empdb;  
hive> SHOW TABLES 'emp.\*';  
employees

**Alter Table**

#### Rename Table

#### Syntax

**ALTER TABLE table\_name RENAME TO new\_table\_name;**

**Example**

**hive> ALTER TABLE employees RENAME TO emp\_sal**

#### Alter Table Properties

**Syntax**

ALTER TABLE table\_name SET TBLPROPERTIES table\_properties;

table\_properties:

  : (property\_name = property\_value, property\_name = property\_value, ... )

**Example**

**ALTER TABLE emp\_sal SET TBLPROPERTIES(‘transactional’ = ‘true’);**

### Drop Table

### Syntax

DROP TABLE [IF EXISTS] table\_name [PURGE];

DROP TABLE removes metadata and data for this table. The data is actually moved to the .Trash/Current directory if Trash is configured (and PURGE is not specified). The metadata is completely lost.

When dropping an EXTERNAL table, data in the table will NOT be deleted from the file system.

When dropping a table referenced by views, no warning is given (the views are left dangling as invalid and must be dropped or recreated by the user).

If PURGE is specified, the table data does not go to the .Trash/Current directory and so cannot be retrieved in the event of a mistaken DROP. The purge option can also be specified with the table property auto.purge.

**Example**

**hive> DROP TABLE acadgild\_big\_data;**

### Truncate Table

### Syntax

TRUNCATE TABLE table\_name [PARTITION partition\_spec];

partition\_spec:

  : (partition\_column = partition\_col\_value, partition\_column = partition\_col\_value, ...)

It removes all rows from a table or partition(s). The rows will be trashed if the filesystem Trash is enabled, otherwise they are deleted.

**Example**

**hive> TRUNCATE TABLE acadgild\_big\_data\_new;**

1. **HiveQL Data Manipulation**

Hive data manipulation language is used to put data into tables and to extract data from tables to the filesystem

Manipulating data is the process of exchanging, moving, sorting, and transforming the data. This technique is used in many situations, such as cleaning data, searching patterns, creating trends, and so on. Hive offers various query statements, keywords, operators, and functions to carry out data manipulation.

HiveQL DM statements are as follows:

* LOAD
* INSERT
* into Hive tables from queries
* into directories from queries
* into Hive tables from SQL
* UPDATE
* DELETE
* MERGE
* EXPORT
* IMPORT

### Loading files into tables

Hive does not do any transformation while loading data into tables. Load operations are currently pure copy/move operations that move datafiles into locations corresponding to Hive tables.

##### **Syntax**

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]

**Example 1:**  **Loading Data from local file system**

**hive> load data local inpath ‘/home/acadgild/emp.txt’  into emp\_sal;**

**Example 2: Loading data from HDFS**

**hive> load data inpath ‘/user/hive/myTables/emp.txt’  into emp\_sal;**

**Inserting data into Hive Tables from queries**

Query Results can be inserted into tables by using the insert clause.

**Syntax**

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select\_statement1 FROM from\_statement;

INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1 FROM from\_statement;

**Hive extension (multiple inserts):**

FROM from\_statement

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select\_statement1

[INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]] select\_statement2]

[INSERT INTO TABLE tablename2 [PARTITION ...] select\_statement2] ...;

FROM from\_statement

INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1

[INSERT INTO TABLE tablename2 [PARTITION ...] select\_statement2]

[INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]] select\_statement2] ...;

**Hive extension (dynamic partition inserts):**

INSERT OVERWRITE TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...) select\_statement FROM from\_statement;

INSERT INTO TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...) select\_statement FROM from\_statement;

**Example**

hive> FROM emp\_sal e

> INSERT OVERWRITE TABLE emp\_1

> SELECT e.id,e.name;

> INSERT OVERWRITE TABLE emp\_2

> SELECT e.salary,e.dept;

**Writing data into the filesystem from queries**

Query results can be inserted into filesystem directories by using a slight variation of the syntax above:

**Syntax**

INSERT OVERWRITE [LOCAL] DIRECTORY directory1

  [ROW FORMAT row\_format] [STORED AS file\_format]

  SELECT ... FROM ...

**Hive extension (multiple inserts):**

FROM from\_statement

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select\_statement1

[INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select\_statement2] ...

**row\_format:**

DELIMITED

[FIELDS TERMINATED BY char [ESCAPED BY char]] [COLLECTION ITEMS TERMINATED BY char]

 [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]

 [NULL DEFINED AS char]

**Example**

INSERT OVERWRITE LOCAL DIRECTORY /home/acadgild/hive/mydata

  ROW FORMAT DELIMITED STORED AS RCFILE

  SELECT \* from emp\_sal;

**Inserting values into tables from SQL**

The INSERT...VALUES statement can be used to insert data into tables directly from SQL.

**Syntax**

INSERT INTO TABLE tablename [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES values\_row [, values\_row ...];

Where values\_row is:

( value [, value ...] )

where a value is either null or any valid SQL literal

**Example**

Hive> INSERT INTO TABLE emp\_sal VALUES (1, ‘deepika’, 50000, ‘Technical’),(2, ‘parshant’, 60000, ‘Management’);

**Update**

**NOTE: Update can only be performed on tables that support ACID.**

**Syntax**

UPDATE tablename SET column = value [, column = value ...] [WHERE expression]; **Example**

hive> UPDATE emp\_sal SET salary = 55000 where name = ‘deepika’;

**Delete**

**NOTE: Delete can only be performed on tables that support ACID.**

**Syntax**DELETE FROM tablename [WHERE expression] **Example**

hive> DELETE FROM emp\_sal where salary = 55000;

**Merge**

**NOTE: Merge can only be performed on tables that support ACID.**

**Syntax**

MERGE INTO <target table> AS T USING <source expression/table> AS S

ON <boolean expression1>

WHEN MATCHED [AND <boolean expression2>] THEN UPDATE SET <set clause list>

WHEN MATCHED [AND <boolean expression3>] THEN DELETE

WHEN NOT MATCHED [AND <boolean expression4>] THEN INSERT VALUES<value list>

**Example**

MERGE INTO emp\_sal\_new AS e1 USING emp\_sal AS e2

ON salary>50000

WHEN MATCHED THEN UPDATE SET dept = ‘Top Management’

WHEN NOT MATCHED THEN INSERT VALUES (1, ‘deepika’, 58000, ‘Technical’);

1. **HiveQL Queries**

HiveQL Queries statements are as follows:

* HiveQL select...where
* HiveQL select...order by
* HiveQL select...group by
* HiveQL select...join

**Syntax of select.... statement**

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];

**“employee table”**

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

| 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 |

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

**HiveQL select...where**

**Example**

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

On successful execution of the query, following output is displayed:

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

| 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**

**Example**

hive> SELECT \* FROM employee ORDER BY DEPT;

On successful execution of the query, following output is displayed:

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

| 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 |

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

**HiveQL select...group by**

**Example**

hive> SELECT Dept,count(\*) as totalEmployees FROM employee GROUP BY DEPT;

On successful execution of the query, following output is displayed:

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

| Dept | totalEmployees |

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

|Admin | 1 |

|HR | 1 |

|PR | 1 |

|TP | 2 |

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

**HiveQL select...join**

**“emp1” table**

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

| ID | Name | Salary |

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

|1201 | Gopal | 45000 |

|1202 | Manisha | 45000 |

|1203 | Masthanvali | 40000 |

|1204 | Krian | 40000 |

|1205 | Kranthi | 30000 |

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

**“emp2” table**

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

| ID | Designation | Dept |

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

|1201 | Technical manager | TP |

|1202 | Proofreader | PR |

|1203 | Technical writer | TP |

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

**Example**

hive> SELECT e1.ID, e1.Name, e2.Designation, e2.Dept From emp1 e1 LEFT OUTER JOIN emp2 e2

On e1.ID = e2.ID;

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

| ID | Name | Designation | Dept |

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

|1201 | Gopal | Technical Manager | TP |

|1202 | Manisha | Proofreader | PR |

|1203 | Masthanvali | Technical writer | TP |

|1204 | Krian | null | null |

|1205 | Kranthi | null | null |

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