Hive commands.

Create Database

- This command will create a database.

```
hive> create database <database-name>;
For Ex - create database demo;
```

- This command will show all databases that are present.

```
hive> show databases;
```

- This command will only create a database if it is not present.

```
hive> create database if not exists <database-name>;
For ex - create database if not exists demo;
```

- Assigning properties with the database in the form of key-value pair.

```
hive> create the database demo
    > WITH DBPROPERTIES ('creator'='Varchasa Aggarwal',
'date'='18-04-2021');
```

- Let's retrieve the information associated with the database.

```
hive> describe database extended demo;
```

Drop Database

Delete a defined database.

- This command will delete a database.

```
hive> drop database demo;
```

- To check that database is deleted or not.

```
hive> show databases;
```

- Drop database if and only if it exists.

```
hive> drop database if exists demo;
```

- In Hive, it is not allowed to drop the database that contains the tables directly. In such a case, we can drop the database either by dropping tables first or use Cascade keyword with the command.
- Let's see the cascade command used to drop the database:-

```
hive> drop database if exists demo cascade.
```

This command automatically drops the table present in the database first.

Hive — Create Table

In hive, we have two types of table —

- Internal table
- External table

Internal Table

- The internal tables are also called managed tables as the lifecycle of their data is controlled by the Hive.
- By default, these tables are stored in a subdirectory under the directory defined by hive.metastore.warehouse.dir (i.e. /user/hive/warehouse).

```
hive> create table demo.employee(Id int, Name string, Salary
float)
    > row format delimited
    > fields terminated by ',';
```

Here, the command also includes the information that the data is separated by ‘,’.

- Let’s see the metadata of the created table.

```
hive> describe demo.employee
```

- Let’s create a table if it not exists.

```
hive> create table if not exists demo.employee(Id int, Name
string, Salary float)
    > row format delimited
    > fields terminated by ','
```

- While creating a table, we can add the comments to the columns and can also define the table properties.

```
hive> create table demo.new_employee(Id int comment 'Employee
Id' Name string comment 'Employee Name', Salary float comment
'Employee Salary') comment 'Table Description' TBLProperties
('creator'='Varchasa Aggarwal','created at'='18-04-2021');
```

- Let's see the metadata of the created table.

```
hive> describe new_employee;
```

- Hive allows creating a new table by using the schema of an existing table.

Schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated.

```
hive> create table if not exists demo.copy_employee like
demo.employee;
```

Here, we can say that the new table is a copy of an existing table.

Hive — Load Data

Once the internal table has been created, the next step is to load the data into it.

- Let's load the data of the file into the database by using the following command: -

```
load data local inpath '/home/<username>/hive/emp_details'into
table demo.employee;select * from.demo.employee;
```

Hive — Drop Table

Let's delete a specific table from the database.

```
hive> show databases;
hive> use demo;
hive> show tables;
hive> drop table new_employee;
hive> show tables;
```

Hive — Alter table

In Hive, we can perform modifications in the existing table like changing the table name, column name, comments, and table properties.

- Rename a table

```
hive> Alter table <old_table_name> rename to <new_table_name>
```

Let's check table name changed or not.

```
hive> show tables;
```

- Adding a column —

```
Alter table table_name add columns(columnName datatype);
```

- Change column —

```
hive> Alter table_name change <old_column_name>
<new_column_name> datatype;
```

- Delete or replace column —

```
alter table employee_data replace columns( id string, first_name
string, age int);
```

In hive with DML statements, we **can add data to the Hive table in 2 different ways**.

- Using INSERT Command
- Load Data Statement

1. Using INSERT Command

Syntax:

```
INSERT INTO TABLE <table_name> VALUES (<add values as per column
entity>);
```

Example:

To insert data into the table let's create a table with the name ***student*** (By default hive uses its *default* database to store hive tables).

Command:

```
CREATE TABLE IF NOT EXISTS student(

Student_Name STRING,

Student_Rollno INT,

Student_Marks FLOAT)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';
```

```
hive> CREATE TABLE IF NOT EXISTS student(
    > Student_Name STRING,
    > Student_Rollno INT,
    > Student_Marks FLOAT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 2.086 seconds
hive> show tables in default;
OK
student
Time taken: 0.268 seconds, Fetched: 1 row(s)
hive>
```

We have successfully created the *student* table in the Hive *default* database with the attribute *Student_Name*, *Student_Rollno*, and *Student_Marks* respectively.

Now, let's insert data into this table with an INSERT query.

INSERT Query:

```
INSERT INTO TABLE student VALUES ('Dikshant',1,'95'),('Akshat', 2 , '96'),('Dhruv',3,'90');
```

```
hive> INSERT INTO TABLE student VALUES ('Dikshant',1,'95'),('Akshat', 2 , '96'),
('Dhruv',3,'90');
Query ID = dikshant_20201106121659_f5dfa694-f552-4b7a-a64b-4f3804213ab8
Total jobs = 3
Launching Job 1 out of 3
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:
```

We can check the data of the ***student*** table with the help of the below command.

```
SELECT * FROM student;
```

```
hive> SELECT * FROM student;
OK
Dikshant        1       95.0
Akshat  2       96.0
Dhruv   3       90.0
Time taken: 0.162 seconds, Fetched: 3 row(s)
hive>
```

2. Load Data Statement

Hive provides us the functionality to load pre-created table entities either from our local file system or from HDFS. The *LOAD DATA* statement is used to load data into the hive table.

Syntax:

```
LOAD DATA [LOCAL] INPATH '<The table data location>' [OVERWRITE]
INTO TABLE <table_name>;
```

Note:

- The *LOCAL* Switch specifies that the data we are loading is available in our Local File System. If the *LOCAL* switch is not used, the hive will consider the location as an HDFS path location.
- The OVERWRITE switch allows us to overwrite the table data.

Let's make a CSV(Comma Separated Values) file with the name *data.csv* since we have provided ',' as a field terminator while creating a table in the hive. We are creating this file in our local file system at '*/home/dikshant/Documents'* for demonstration purposes.

Command:

```
cd /home/dikshant/Documents    // To change the directory

touch data.csv                 // use to create data.csv file

nano data.csv                  // nano is a linux command line
editor to edit files

cat data.csv                   // cat is used to see content of
file
```

```
dikshant@dikshant:~$ cd /home/dikshant/Documents/
dikshant@dikshant:~/Documents$ touch data.csv
dikshant@dikshant:~/Documents$ nano data.csv
dikshant@dikshant:~/Documents$ cat data.csv
Ganesh,4,85
Chandan,5,65
Bhavani,6,87
dikshant@dikshant:~/Documents$
```

LOAD DATA to the student hive table with the help of the below command.

```
LOAD DATA LOCAL INPATH '/home/dikshant/Documents/data.csv' INTO
TABLE student;
```

```
hive> LOAD DATA LOCAL INPATH '/home/dikshant/Documents/data.csv' INTO TABLE stud
ent;
Loading data to table default.student
OK
Time taken: 2.617 seconds
hive>
```

Let's see the *student* table content to observe the effect with the help of the below command.

```
SELECT * FROM student;
```

```
hive> SELECT * FROM student;
OK
Dikshant        1       95.0
Akshat  2       96.0
Dhruv   3       90.0
Ganesh  4       85.0
Chandan 5       65.0
Bhavani 6       87.0
Time taken: 1.24 seconds, Fetched: 6 row(s)
hive>
```

We can observe that we have successfully added the data to the *student* table.

Hive — Partitioning

The partitioning in hive can be done in two ways —

- Static partitioning
- Dynamic partitioning

Static Partitioning

In static or manual partitioning, it is required to pass the values of partitioned columns manually while loading the data into the table. Hence, the data file doesn't contain the partitioned columns.

```
hive> use test;
hive> create table student (id int, name string, age int,
institute string)
    > partitioned by (course string)
    > row format delimited
    > fields terminated by ',';
```

- Let's retrieve the information.

```
hive> describe student;
```

- Load the data into the table and pass the values of partition columns with it by using the following command: -

```
hive> load data local inpath
'/home/<username>/hive/student_details1' into table student
```

```
partition(course= "python");hive> load data local inpath
'/home/<username>/hive/student_details1' into table student
partition(course= "Hadoop");
```

- Now retrieve the data.

```
hive> select * from student;
hive> select * from student where course = 'Hadoop';
```

Dynamic Partitioning

In dynamic partitioning, the values of partitioned columns exist within the table. So, it is not required to pass the values of partitioned columns manually.

```
hive> use show;
```

- Enable the dynamic partitioning.

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

- Create the dummy table.

```
hive> create table stud_demo(id int, name string, age int,
institute string, course string)
row format delimited
fields terminated by ',';
```

- Now load the data.

```
hive> load data local inpath
'/home/<username>/hive/student_details' into table stud_demo;
```

- Create a partition table.

```
hive> create table student_part (id int, name string, age int,
institute string)
partitioned by (course string)
row format delimited
fields terminated by ',';
```

- Insert the data of dummy table in the partition table.

```
hive> insert into student_part
partition(course)
select id, name age, institute, course
```

```
from stud_demo;
```

- Now you can view the table data with the help of *select* command.

HiveQL — Operators

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

```
hive> use hql;
hive> create table employee (Id int, Name string , Salary float)
row format delimited
fields terminated by ',' ;
```

- Now load the data.

```
hive> load data local inpath '/home/<username>/hive/emp_data'
into table employee;
```

- Fetch the data.

```
select * from employee;
```

Arithmetic Operators in Hive

Operators	Description
A + B	This is used to add A and B.
A - B	This is used to subtract B from A.
A * B	This is used to multiply A and B.
A / B	This is used to divide A and B and returns the quotient of the operands.
A % B	This returns the remainder of A / B.
A | B	This is used to determine the bitwise OR of A and B.
A & B	This is used to determine the bitwise AND of A and B.
A ^ B	This is used to determine the bitwise XOR of A and B.
~A	This is used to determine the bitwise NOT of A.

- Adding 50 to salary column.

```
hive> select id, name, salary + 50 from employee;
```

- Substracting 50 from the salary column.

```
hive> select id, name, salary -50 from employee;
```

- Find out the 10% salary of each employee.

```
hive> select id, name, salary *10 from employee;
```

Relational Operators in Hive

Operator	Description
A=B	It returns true if A equals B, otherwise false.
A <> B, A !=B	It returns null if A or B is null; true if A is not equal to B, otherwise false.
A<B	It returns null if A or B is null; true if A is less than B, otherwise false.
A>B	It returns null if A or B is null; true if A is greater than B, otherwise false.
A<=B	It returns null if A or B is null; true if A is less than or equal to B, otherwise false.
A>=B	It returns null if A or B is null; true if A is greater than or equal to B, otherwise false.
A IS NULL	It returns true if A evaluates to null, otherwise false.
A IS NOT NULL	It returns false if A evaluates to null, otherwise true.

- Fetch the details of the employee having salary>=25000.

```
hive> select * from employee where salary>=25000;
```

- Fetch the details of the employee having salary<25000.

```
hive> select * from employee where salary < 25000;
```

Functions in Hive

```
hive> use hql;
hive> create table employee_data (Id int, Name string , Salary
float)
row format delimited
fields terminated by ',' ;
```

- Now load the data.

```
hive> load data local inpath
'/home/<username>/hive/employee_data' into table employee;
```

- Fetch the data.

```
select * from employee_data;
```

Mathematical Functions in Hive

Return type	Functions	Description
BIGINT	round(num)	It returns the BIGINT for the rounded value of DOUBLE num.
BIGINT	floor(num)	It returns the largest BIGINT that is less than or equal to num.
BIGINT	ceil(num), ceiling(DOUBLE num)	It returns the smallest BIGINT that is greater than or equal to num.
DOUBLE	exp(num)	It returns exponential of num.
DOUBLE	ln(num)	It returns the natural logarithm of num.
DOUBLE	log10(num)	It returns the base-10 logarithm of num.
DOUBLE	sqrt(num)	It returns the square root of num.
DOUBLE	abs(num)	It returns the absolute value of num.
DOUBLE	sin(d)	It returns the sin of num, in radians.
DOUBLE	asin(d)	It returns the arcsin of num, in radians.
DOUBLE	cos(d)	It returns the cosine of num, in radians.
DOUBLE	acos(d)	It returns the arccosine of num, in radians.
DOUBLE	tan(d)	It returns the tangent of num, in radians.
DOUBLE	atan(d)	It returns the arctangent of num, in radians.

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

```
hive> select Id, Name, sqrt(Salary) from employee_data ;
```

Aggregate Functions

Return Type	Operator	Description
BIGINT	count(*)	It returns the count of the number of rows present in the file.
DOUBLE	sum(col)	It returns the sum of values.
DOUBLE	sum(DISTINCT col)	It returns the sum of distinct values.
DOUBLE	avg(col)	It returns the average of values.
DOUBLE	avg(DISTINCT col)	It returns the average of distinct values.
DOUBLE	min(col)	It compares the values and returns the minimum one form it.
DOUBLE	max(col)	It compares the values and returns the maximum one form it.

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

```
hive> select max(Salary) from employee_data;
hive> select min(Salary) from employee_data;
```

Other functions in Hive

Return Type	Operator	Description
INT	length(str)	It returns the length of the string.
STRING	reverse(str)	It returns the string in reverse order.
STRING	concat(str1, str2, ...)	It returns the concatenation of two or more strings.
STRING	substr(str, start_index)	It returns the substring from the string based on the provided starting index.
STRING	substr(str, int start, int length)	It returns the substring from the string based on the provided starting index and length.
STRING	upper(str)	It returns the string in uppercase.
STRING	lower(str)	It returns the string in lowercase.
STRING	trim(str)	It returns the string by removing whitespaces from both the ends.
STRING	ltrim(str)	It returns the string by removing whitespaces from left-hand side.
TRING	rtrim(str)	It returns the string by removing whitespaces from right-hand side.

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

```
hive> select Id, upper(Name) from employee_data;
```

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

```
hive> select Id, lower(Name) from employee_data;
```

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.

```
hive> use hql;
hive> create table employee_data (Id int, Name string , Salary
float)
row format delimited
fields terminated by ',' ;
```

- Now load the data.

```
hive> load data local inpath
'/home/<username>/hive/employee_data' into table employee;
```

- Fetch the data.

```
select department, sum(salary) from employee_data group by
department;
```

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**.

- Let's fetch the sum of employee's salary based on department having sum >= 35000 by using the following command:

```
hive> select department, sum(salary) from emp group by
department having sum(salary)>=35000;
```

HiveQL — 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.

```
hive> use hql;
hive> create table employee_data (Id int, Name string , Salary
float)
row format delimited
fields terminated by ',' ;
```

- Now load the data.

```
hive> load data local inpath
'/home/<username>/hive/employee_data' into table employee;
```

- Fetch the data.

```
select * from emp order by salary desc;
```

HiveQL — 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 order by salary desc;
```