Hive Data Definition Language. **Hive Data Definition Language** (**DDL**) is a subset of **Hive** SQL statements that describe the **data** structure in **Hive** by creating, deleting, or altering schema objects such as databases, tables, views, partitions, and buckets.

Hive DDL Commands

← prev next →

Create Database Statement

A database in Hive is a namespace or a collection of tables.

```
hive> CREATE SCHEMA userdb;
hive> SHOW DATABASES;
```

Drop database

```
hive> DROP DATABASE IF EXISTS userdb;
```

Creating Hive Tables

Create a table called Sonoo with two columns, the first being an integer and the other a string.

```
hive> CREATE TABLE Sonoo(foo INT, bar STRING);
```

Create a table called HIVE_TABLE with two columns and a partition column called ds. The partition column is a virtual column. It is not part of the data itself but is derived from the partition that a particular dataset is loaded into.By default, tables are assumed to be of text input format and the delimiters are assumed to be ^A(ctrl-a).

```
hive> CREATE TABLE HIVE_TABLE (foo INT, bar STRING) PARTITIONED BY (ds STRING);
```

Browse the table

```
hive>  Show tables;
```

Altering and Dropping Tables

```
hive> ALTER TABLE Sonoo RENAME TO Kafka;
hive> ALTER TABLE Kafka ADD COLUMNS (col INT);
hive> ALTER TABLE HIVE_TABLE ADD COLUMNS (col1 INT COMMENT 'a comment');
hive> ALTER TABLE HIVE_TABLE REPLACE COLUMNS (col2 INT, weight STRING, baz INT COMMENT 'baz replaces new_col1');
```

DML refers to "Data Manipulation Language", a subset of SQL statements that modify the data stored in tables.

Hive DML Commands

← prev next →

To understand the Hive DML commands, let's see the employee and employee_department table first.

Employee			Employee Department	
EMP ID	Emp Name	Address	Emp ID	Department
1	Rose	US	1	IT
2	Fred	US	2	IT
3	Jess	In	3	Eng
4	Frey	Th	4	Admin

LOAD DATA

```
hive> LOAD DATA LOCAL INPATH './usr/Desktop/kv1.txt' OVERWRITE INTO TABLE Employee;
```

SELECTS and FILTERS

```
hive> SELECT  E.EMP_ID FROM Employee E  WHERE E.Address='US';
```

GROUP BY

```
hive> hive> SELECT  E.EMP_ID FROM Employee E GROUP BY E.Addresss;
```

There are multiple ways to modify data in Hive:

- LOAD
- INSERT
  - into Hive tables from queries
  - into directories from queries
  - into Hive tables from SQL
- UPDATE
- DELETE
- MERGE

HiveQL(Hive Query Language)

Hive provides a command line interface (CLI). Hive provides a CLI to write Hive queries using Hive Query Language (Hive-QL).

Generally, HiveQL syntax is similar to the SQL syntax that most data analysts are familiar with. Hive supports four file formats those are TEXTFILE, SEQUENCEFILE, ORC and RCFILE (Record Columnar File).

- For single user meta data storage Hive uses derby database and
- For multiple user Meta data or shared Meta data case Hive uses MYSQL

Built-in operators

Hive provides Built-in operators for Data operations to be implemented on the tables present inside Hive warehouse.

These operators are used for mathematical operations on operands, and it will return specific value as per the logic applied.

Types of Built-in Operators in HIVE are:

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Operators on Complex types
- Complex type Constructors

Relational Operators:

We use Relational operators for relationship comparisons between two operands.

- Operators such as equals, Not equals, less than, greater than …etc
- The operand types are all number types in these Operators.

The following Table will give us details about Relational operators and its usage.

Built-in Operator	Description	Operand
X = Y	TRUE if expression X is equivalent to expression Y Otherwise FALSE.	It takes all primitive types
X != Y	TRUE if expression X is not equivalent to expression Y Otherwise FALSE.	It takes all primitive types
X < Y	TRUE if expression X is less than expression Y Otherwise FALSE.	It takes all primitive types

Arithmetic Operators:

The following Table will give us details about Arithmetic operators

Built-in Operator	Description	Operand
X + Y	It will return the output of adding X and Y value.	It takes all number types
X - Y	It will return the output of subtracting Y from X value.	It takes all number types
X * Y	It will return the output of multiplying X and Y values.	It takes all number types
X / Y	It will return the output of dividing Y from X.	It takes all number types
X % Y	It will return the remainder resulting from dividing X by Y.	It takes all number types
X & Y	It will return the output of bitwise AND of X and Y.	It takes all number types
X | Y	It will return the output of bitwise OR of X and Y.	It takes all number types
X ^ Y	It will return the output of bitwise XOR of X and Y.	It takes all number types
~X	It will return the output of bitwise NOT of X.	It takes all number types

Logical Operators:

We use Logical operators for performing Logical operations on operands

- Logical operations such as AND, OR, NOT between operands we use these Operators.
- The operand types all are BOOLEAN type in these Operators

The following Table will give us details about Logical operators

Operators	Description	Operands
X AND Y	TRUE if both X and Y are TRUE, otherwise FALSE.	Boolean types only
X && Y	Same as X AND Y but here we using && symbol	Boolean types only
X OR Y	TRUE if either X or Y or both are TRUE, otherwise FALSE.	Boolean types only
X || Y	Same as X OR Y but here we using || symbol	Boolean types only
NOT X	TRUE if X is FALSE, otherwise FALSE.	Boolean types only
!X	Same as NOT X but here we using! symbol	Boolean types only

Complex type Constructors:

The following Table will give us details about Complex type Constructors. It will construct instances on complex data types. These are of complex data types such as Array, Map and Struct types in Hive.

In this section, we are going to see the operations performed on Complex type Constructors.

Operators	Operands	Description
array	(val1, val2, ...)	It will create an array with the given elements as mentioned like val1, val2
Create_ union	(tag, val1, val2, ...)	It will create a union type with the values that is being mentioned to by the tag parameter
map	(key1, value1, key2, value2, ...)	It will create a map with the given key/value pairs mentioned in operands
Named_struct	(name1, val1, name2, val2, ...)	It will create a Struct with the given field names and values mentioned in operands
STRUCT	(val1, val2, val3, ...)	Creates a Struct with the given field values. Struct field names will be col1, col2, .