**Hive Data Definitions**

Hive Data Definition language (DDL) is used to create, alter and drop the schema/databases and table.

HiveQL DDL statements are documented here, including:

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

**Create Database Statement:**

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

*Example:*

CREATE SCHEMA/DATABASE IF NOT EXISTS userdb;

SHOW DATABASES;

**Drop database:** This command will delete the entire database**.**

*Example:*

DROP DATABASE IF EXISTS userdb;

**Creating Hive Tables**

**Managed and External Tables**

By default Hive creates managed tables, where files, metadata and statistics are managed by internal Hive processes. A managed table is stored under the [hive.metastore.warehouse.dir](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties#ConfigurationProperties-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.

Use managed tables when Hive should manage the lifecycle of the table, or when generating temporary tables.

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 location

Use external tables when files are already present or in remote locations, and the files should remain even if the table is dropped.

Managed or external tables can be identified using the [DESCRIBE FORMATTED table\_name](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DescribeTable/View/Column) command, which will display either MANAGED\_TABLE or EXTERNAL\_TABLE depending on table type.

**Managed Table:**

Create a managed table called Acad1 with two columns, the first being an integer and the other a string.

*Example:*

CREATE TABLE Acad1(foo INT, bar STRING);   --🡪 Managed Table.

**External Tables:**

An External table can be created by using Key Word EXTERNAL

Eg:

CREATE EXTERNAL TABLE page\_view(viewTime INT, userid BIGINT,

     page\_url STRING, referrer\_url STRING,

     ip STRING COMMENT 'IP Address of the User',

     country STRING COMMENT 'country of origination')

 COMMENT 'This is the staging page view table'

 ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054'

 STORED AS TEXTFILE

 LOCATION '<hdfs\_location>';

*Example:*

**Browse the table:**

Show tables;

#### 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.

Partitioned Table based on ds column as partition key.

*Example:*

CREATE TABLE HIVE\_TABLE (foo INT, bar STRING) PARTITIONED BY (ds 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 ‘ ,’ .

**Bucketed Tables:**

Bucketed tables are fantastic in that they allow much more efficient [sampling](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling) than do non-bucketed tables, and they may later allow for time saving operations such as mapside joins

*Example:*

|  |
| --- |
| CREATE TABLE user\_info\_bucketed(user\_id BIGINT, firstname STRING, lastname STRING)  COMMENT 'A bucketed copy of user\_info'  PARTITIONED BY(ds STRING)  CLUSTERED BY(user\_id) INTO 256 BUCKETS; |

In the example above, the user\_info table is bucketed (clustered by) userid and within each bucket the data is sorted in increasing order of viewTime. Such an organization allows the user to do efficient sampling on the clustered column - in this case userid. The clustering property allows internal operators to take advantage of the better-known data structure while evaluating queries, also increasing efficiency.

The CLUSTERED BY creation commands do not affect how data is inserted into a table – only how it is read. This means that users must be careful to insert data correctly by specifying the number of reducers to be equal to the number of buckets, and using CLUSTER BY commands in their query.

**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](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select) 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 new\_key\_value\_store  ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"  STORED AS RCFile  AS  SELECT (key % 1024) new\_key, concat(key, value) key\_value\_pair  FROM key\_value\_store  SORT BY new\_key, key\_value\_pair; |

The above CTAS statement creates the target table new\_key\_value\_store with the schema (new\_key DOUBLE, key\_value\_pair 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.

Starting with [Hive 0.13.0](https://issues.apache.org/jira/browse/HIVE-1180), the SELECT statement can include one or more common table expressions (CTEs), as shown in the [SELECT syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-SelectSyntax). For an example, see [Common Table Expression](https://cwiki.apache.org/confluence/display/Hive/Common+Table+Expression#CommonTableExpression-CTEinViews,CTAS,andInsertStatements).

Being able to select data from one table to another is one of the most powerful features of Hive. Hive handles the conversion of the data from the source format to the destination format as the query is being executed.

**Create Table Like**

The LIKE form of CREATE TABLE allows you to copy an existing table definition exactly (without copying its data). In contrast to CTAS, the statement below creates a new empty\_key\_value\_store table whose definition exactly matches the existing key\_value\_store in all particulars other than table name. The new table contains no rows.

|  |
| --- |
| *Example:*  CREATE TABLE empty\_key\_value\_store  LIKE key\_value\_store; |

**Altering and Dropping Tables**

*Example:*

ALTER TABLE Acad1 RENAME TO Kafka;

ALTER TABLE Kafka ADD COLUMNS (col INT);

ALTER TABLE HIVE\_TABLE ADD COLUMNS (col1 INT COMMENT 'a comment');

ALTER TABLE HIVE\_TABLE REPLACE COLUMNS (col2 INT, weight STRING, baz INT COMMENT 'baz replaces new\_col1');

DROP TABLE [IF EXISTS] Acad1;

Drop command will delete the entire table data along with its schema.