**External Tables**

The EXTERNAL keyword lets you create a table and provide a LOCATION so that Hive does not use a default location for this table. This comes in handy if you already have data generated. When dropping 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](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties#ConfigurationProperties-hive.metastore.warehouse.dir).

**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.

**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.

**Temporary Tables**

A table that has been created as a temporary table will only be visible to the current session. Data will be stored in the user's scratch directory, and deleted at the end of the session.

If a temporary table is created with a database/table name of a permanent table which already exists in the database, then within that session any references to that table will resolve to the temporary table, rather than to the permanent table. The user will not be able to access the original table within that session without either dropping the temporary table, or renaming it to a non-conflicting name.

Temporary tables have the following limitations:

* Partition columns are not supported.
* No support for creation of indexes.

**Drop Table**

**Temporary Tables**

A table that has been created as a temporary table will only be visible to the current session.

Data will be stored in the user's scratch directory, and deleted at the end of the session.

If a temporary table is created with a database/table name of a permanent table which already exists in the database, then within that session any references to that table will resolve to the temporary table

Temporary tables have the following limitations:

* Partition columns are not supported.
* No support for creation of indexes.

Starting in [Hive 1.1.0](https://issues.apache.org/jira/browse/HIVE-7313) the storage policy for temporary tables can be set to memory, ssd, or default with the [hive.exec.temporary.table.storage](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties" \l "ConfigurationProperties-hive.exec.temporary.table.storage) configuration parameter

### Truncate Table

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

Currently the target table should be native/managed table or an exception will be thrown

User can specify partial partition\_spec for truncating multiple partitions at once and omitting partition\_spec will truncate all partitions in the table.

Starting with HIVE 2.3.0 ([HIVE-15880](https://issues.apache.org/jira/browse/HIVE-15880)) if the table property "auto.purge" (see [TBLPROPERTIES](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-listTableProperties) above) is set to "true" the data of the table is not moved to Trash when a TRUNCATE TABLE command is issued against it and cannot be retrieved in the event of a mistaken TRUNCATE.

### Alter Table

#### Rename Table

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

As of version 0.6, a rename on a [managed table](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-managedTable) moves its HDFS location. Rename has been changed as of version 2.2.0 ([HIVE-14909](https://issues.apache.org/jira/browse/HIVE-14909)) so that a managed table's HDFS location is moved only if the table is created without a [LOCATION clause](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateTable) and under its database directory. Hive versions prior to 0.6 just renamed the table in the metastore without moving the HDFS location.

#### Alter Table Properties

ALTER TABLE table\_name SET TBLPROPERTIES table\_properties;

table\_properties:

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

You can use this statement to add your own metadata to the tables. Currently last\_modified\_user, last\_modified\_time properties are automatically added and managed by Hive. Users can add their own properties to this list. You can do DESCRIBE EXTENDED TABLE to get this information.

##### Alter Table Comment

To change the comment of a table you have to change the comment property of the TBLPROPERTIES:

ALTER TABLE table\_name SET TBLPROPERTIES ('comment' = new\_comment);

#### Add SerDe Properties

ALTER TABLE table\_name [PARTITION partition\_spec] SET SERDE serde\_class\_name [WITH SERDEPROPERTIES serde\_properties];

ALTER TABLE table\_name SET SERDEPROPERTIES ('field.delim' = ',');

##### Alter Table Skewed

The STORED AS DIRECTORIES option determines whether a [skewed](https://cwiki.apache.org/confluence/display/Hive/Skewed+Join+Optimization) table uses the [list bucketing](https://cwiki.apache.org/confluence/display/Hive/ListBucketing) feature, which creates subdirectories for skewed values.

##### Alter Table Not Skewed

ALTER TABLE table\_name NOT SKEWED;

The NOT SKEWED option makes the table non-skewed and turns off the list bucketing feature (since a list-bucketing table is always skewed). This affects partitions created after the ALTER statement, but has no effect on partitions created before the ALTER statement.

##### Alter Table Not Stored as Directories

|  |
| --- |
| ALTER TABLE table\_name NOT STORED AS DIRECTORIES; |

This turns off the list bucketing feature, although the table remains skewed.

##### Alter Table Set Skewed Location

|  |
| --- |
| ALTER TABLE table\_name SET SKEWED LOCATION (col\_name1="location1" [, col\_name2="location2", ...] ); |

This changes the location map for list bucketing.

#### Alter Table Constraints

ALTER TABLE table\_name ADD CONSTRAINT constraint\_name PRIMARY KEY (column, ...) DISABLE NOVALIDATE;

ALTER TABLE table\_name ADD CONSTRAINT constraint\_name FOREIGN KEY (column, ...) REFERENCES table\_name(column, ...) DISABLE NOVALIDATE RELY;

ALTER TABLE table\_name DROP CONSTRAINT constraint\_name;

The CLUSTERED BY and SORTED 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 and SORT BY commands in their query.

## Overview of Hive Indexes

The goal of Hive indexing is to improve the speed of query lookup on certain columns of a table. Without an index, queries with predicates like 'WHERE tab1.col1 = 10' load the entire table or partition and process all the rows. But if an index exists for col1, then only a portion of the file needs to be loaded and processed.

The improvement in query speed that an index can provide comes at the cost of additional processing to create the index and disk space to store the index.

# Hive HBase Integration

This feature allows Hive QL statements to access [HBase](http://hadoop.apache.org/hbase) tables for both read (SELECT) and write (INSERT). It is even possible to combine access to HBase tables with native Hive tables via joins and unions.

The storage handler is built as an independent module, hive-hbase-handler-x.y.z.jar, which must be available on the Hive client auxpath, along with HBase, Guava and ZooKeeper jars. It also requires the correct configuration property to be set in order to connect to the right HBase master.

# Hive Storage Handlers

The motivation is to make it possible to allow Hive to access data stored and managed by other systems in a modular, extensible fashion

Hive storage handler support builds on existing extensibility features in both Hadoop and Hive:

* input formats
* output formats
* serialization/deserialization libraries

Besides bundling these together, a storage handler can also implement a new metadata hook interface, allowing Hive DDL to be used for managing object definitions in both the Hive metastore and the other system's catalog simultaneously and consistently.

Storage handlers introduce a distinction between native and non-native tables. A native table is one which Hive knows how to manage and access without a storage handler; a non-native table is one which requires a storage handler.

These two distinctions (*managed vs. external* and *native vs non-native*) are orthogonal. Hence, there are four possibilities for base tables:

* managed native: what you get by default with CREATE TABLE
* external native: what you get with CREATE EXTERNAL TABLE when no STORED BY clause is specified
* managed non-native: what you get with CREATE TABLE when a STORED BY clause is specified; Hive stores the definition in its metastore, but does not create any files itself; instead, it calls the storage handler with a request to create a corresponding object structure
* external non-native: what you get with CREATE EXTERNAL TABLE when a STORED BY clause is specified; Hive registers the definition in its metastore and calls the storage handler to check that it matches the primary definition in the other system

When STORED BY is specified, then row\_format (DELIMITED or SERDE) and STORED AS cannot be specified. Optional SERDEPROPERTIES can be specified as part of the STORED BY clause and will be passed to the serde provided by the storage handler.

## Column Mapping

There are two SERDEPROPERTIES that control the mapping of HBase columns to Hive:

* hbase.columns.mapping
* hbase.table.default.storage.type: Can have a value of either string (the default) or binary, this option is only available as of Hive 0.9 and the string behavior is the only one available in earlier versions