

LanguageManual DDL

Created by Confluence Administrator, last modified by Lefty Leverenz on Apr 05, 2016

Hive Data Definition Language

- [Hive Data Definition Language](#)
 - [Overview](#)
 - [Keywords, Non-reserved Keywords and Reserved Keywords](#)
 - [Create/Drop/Alter/Use Database](#)
 - [Create/Drop/Truncate Table](#)
 - [Alter Table/Partition/Column](#)
 - [Create/Drop/Alter View](#)
 - [Create/Drop/Alter Index](#)
 - [Create/Drop Macro](#)
 - [Create/Drop/Reload Function](#)
 - [Create/Drop/Grant/Revoke Roles and Privileges](#)
 - [Show](#)
 - [Describe](#)
- [HCatalog and WebHCat DDL](#)

Overview

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
- MSCK REPAIR TABLE (or ALTER TABLE RECOVER PARTITIONS)
- SHOW DATABASES/SCHEMAS, TABLES, TBLPROPERTIES, 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.

Keywords, Non-reserved Keywords and Reserved Keywords

Keywords

ADD, ADMIN, AFTER, ALL, ALTER, ANALYZE, AND, ARCHIVE, ARRAY, AS, ASC, AUTHORIZATION, BEFORE, BETWEEN, BIGINT, BINARY, BOOLEAN, BOTH, BUCKET, BUCKETS, BY, CASCADE, CASE, CAST, CHANGE, CHAR, CLUSTER, CLUSTERED, CLUSTERSTATUS, COLLECTION, COLUMN, COLUMNS, COMMENT, COMPACT, COMPACTIONS, COMPUTE, CONCATENATE, CONF, CONTINUE, CREATE, CROSS, CUBE, CURRENT, CURRENT_DATE, CURRENT_TIMESTAMP, CURSOR, DATA, DATABASE, DATABASES, DATE, DATETIME, DAY, DBPROPERTIES, DECIMAL, DEFERRED, DEFINED, DELETE, DELIMITED, DEPENDENCY, DESC, DESCRIBE, DIRECTORIES, DIRECTORY, DISABLE, DISTINCT, DISTRIBUTE, DOUBLE, DROP, ELEM_TYPE, ELSE, ENABLE, END, ESCAPED, EXCHANGE, EXCLUSIVE, EXISTS, EXPLAIN, EXPORT, EXTENDED, EXTERNAL, FALSE, FETCH, FIELDS, FILE, FILEFORMAT, FIRST, FLOAT, FOLLOWING, FOR, FORMAT, FORMATTED, FROM, FULL, FUNCTION, FUNCTIONS, GRANT, GROUP, GROUPING, HAVING, HOLD_DDLTIME, HOUR, IDXPROPERTIES, IF, IGNORE, IMPORT, IN, INDEX, INDEXES, INNER, INPATH, INPUTDRIVER, INPUTFORMAT, INSERT, INT, INTERSECT, INTERVAL, INTO, IS, ITEMS, JAR, JOIN, KEYS, KEY_TYPE, LATERAL, LEFT, LESS, LIKE, LIMIT, LINES, LOAD, LOCAL, LOCATION, LOCK, LOCKS, LOGICAL, LONG, MACRO, MAP, MAPJOIN, MATERIALIZED, MINUS, MINUTE, MONTH, MORE, MSCK, NONE, NOSCAN, NOT, NO_DROP, NULL, OF, OFFLINE, ON, OPTION, OR, ORDER, OUT, OUTER, OUTPUTDRIVER, OUTPUTFORMAT, OVER, OVERWRITE, OWNER, PARTIALSCAN, PARTITION, PARTITIONED, PARTITIONS, PERCENT, PLUS, PRECEDING, PRESERVE, PRETTY, PRINCIPALS, PROCEDURE, PROTECTION, PURGE, RANGE, READ, READONLY, READS, REBUILD, RECORDREADER, RECORDWRITER, REDUCE, REGEXP, RELOAD, RENAME, REPAIR, REPLACE, RESTRICT, REVOKE, REWRITE, RIGHT, RLIKE, ROLE, ROLES, ROLLUP, ROW, ROWS, SCHEMA, SCHEMAS, SECOND, SELECT, SEMI, SERDE, SERDEPROPERTIES, SERVER, SET, SETS, SHARED, SHOW, SHOW_DATABASE, SKEWED, SMALLINT, SORT, SORTED, SSL, STATISTICS, STORED, STREAMTABLE, STRING, STRUCT, TABLE, TABLES, TABLESAMPLE, TBLPROPERTIES, TEMPORARY, TERMINATED, THEN, TIMESTAMP, TINYINT, TO, TOUCH, TRANSACTIONS, TRANSFORM, TRIGGER, TRUE, TRUNCATE, UNARCHIVE, UNBOUNDED, UNDO, UNION, UNIONTYPE, UNIQUEJOIN, UNLOCK, UNSET, UNSIGNED, UPDATE, URI, USE, USER, USING, UTC, UTCTIMESTAMP, VALUES, VALUE_TYPE, VARCHAR, VIEW, WHEN, WHERE, WHILE, WINDOW, WITH, YEAR

Non-reserved Keywords

ADD, ADMIN, AFTER, ANALYZE, ARCHIVE, ASC, BEFORE, BUCKET, BUCKETS, CASCADE, CHANGE, CLUSTER, CLUSTERED, CLUSTERSTATUS, COLLECTION, COLUMNS, COMMENT, COMPACT, COMPACTIONS, COMPUTE, CONCATENATE, CONTINUE, DATA, DATABASES, DATETIME, DAY, DBPROPERTIES, DEFERRED, DEFINED, DELIMITED, DEPENDENCY, DESC, DIRECTORIES, DIRECTORY, DISABLE, DISTRIBUTE, ELEM_TYPE, ENABLE, ESCAPED, EXCLUSIVE, EXPLAIN, EXPORT, FIELDS, FILE, FILEFORMAT, FIRST, FORMAT, FORMATTED, FUNCTIONS, HOLD_DDLTIME, HOUR, IDXPROPERTIES, IGNORE, INDEX, INDEXES, INPATH, INPUTDRIVER, INPUTFORMAT, ITEMS, JAR, KEYS, KEY_TYPE, LIMIT, LINES, LOAD, LOCATION, LOCK, LOCKS, LOGICAL, LONG, MAPJOIN, MATERIALIZED, MINUS, MINUTE, MONTH, MSCK, NOSCAN, NO_DROP, OFFLINE, OPTION, OUTPUTDRIVER, OUTPUTFORMAT, OVERWRITE, OWNER, PARTITIONED, PARTITIONS, PLUS, PRETTY, PRINCIPALS, PROTECTION,

PURGE, READ, READONLY, REBUILD, RECORDREADER, RECORDWRITER, REGEXP (Hive 0.x.x and 1.x.x), RELOAD, RENAME, REPAIR, REPLACE, RESTRICT, REWRITE, RLIKE (Hive 0.x.x and 1.x.x), ROLE, ROLES, SCHEMA, SCHEMAS, SECOND, SEMI, SERDE, SERDEPROPERTIES, SERVER, SETS, SHARED, SHOW, SHOW_DATABASE, SKEWED, SORT, SORTED, SSL, STATISTICS, STORED, STREAMTABLE, STRING, STRUCT, TABLES, TBLPROPERTIES, TEMPORARY, TERMINATED, TINYINT, TOUCH, TRANSACTIONS, UNARCHIVE, UNDO, UNIONTYPE, UNLOCK, UNSET, UNSIGNED, URI, USE, UTC, UTCTIMESTAMP, VALUE_TYPE, VIEW, WHILE, YEAR

Version information
REGEXP and RLIKE are non-reserved keywords prior to Hive 2.0.0 and reserved keywords starting in Hive 2.0.0 ([HIVE-11703](#)).

Reserved Keywords

ALL, ALTER, AND, ARRAY, AS, AUTHORIZATION, BETWEEN, BIGINT, BINARY, BOOLEAN, BOTH, BY, CASE, CAST, CHAR, COLUMN, CONF, CREATE, CROSS, CUBE, CURRENT, CURRENT_DATE, CURRENT_TIMESTAMP, CURSOR, DATABASE, DATE, DECIMAL, DELETE, DESCRIBE, DISTINCT, DOUBLE, DROP, ELSE, END, EXCHANGE, EXISTS, EXTENDED, EXTERNAL, FALSE, FETCH, FLOAT, FOLLOWING, FOR, FROM, FULL, FUNCTION, GRANT, GROUP, GROUPING, HAVING, IF, IMPORT, IN, INNER, INSERT, INT, INTERSECT, INTERVAL, INTO, IS, JOIN, LATERAL, LEFT, LESS, LIKE, LOCAL, MACRO, MAP, MORE, NONE, NOT, NULL, OF, ON, OR, ORDER, OUT, OUTER, OVER, PARTIALSCAN, PARTITION, PERCENT, PRECEDING, PRESERVE, PROCEDURE, RANGE, READS, REDUCE, REGEXP (Hive 2.0.0 onward), REVOKE, RIGHT, RLIKE (Hive 2.0.0 onward), ROLLUP, ROW, ROWS, SELECT, SET, SMALLINT, TABLE, TABLESAMPLE, THEN, TIMESTAMP, TO, TRANSFORM, TRIGGER, TRUE, TRUNCATE, UNBOUNDED, UNION, UNIQUEJOIN, UPDATE, USER, USING, VALUES, VARCHAR, WHEN, WHERE, WINDOW, WITH

Reserved keywords are permitted as identifiers if you quote them as described in [Supporting Quoted Identifiers in Column Names](#) (version 0.13.0 and later, see [HIVE-6013](#)). Most of the keywords are reserved through [HIVE-6617](#) in order to reduce the ambiguity in grammar (version 1.2.0 and later). There are two ways if the user still would like to use those reserved keywords as identifiers: (1) use quoted identifiers, (2) set `hive.support.sql11.reserved.keywords=false`.

Create/Drop/Alter/Use Database

Create Database

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
  [COMMENT database_comment]
  [LOCATION hdfs_path]
  [WITH DBPROPERTIES (property_name=property_value, ...)];
```

The uses of SCHEMA and DATABASE are interchangeable – they mean the same thing. CREATE DATABASE was added in Hive 0.6 ([HIVE-675](#)). The WITH DBPROPERTIES clause was added in Hive 0.7 ([HIVE-1836](#)).

Drop Database

```
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];
```

The uses of SCHEMA and DATABASE are interchangeable – they mean the same thing. DROP DATABASE was added in Hive 0.6 ([HIVE-675](#)).

Alter Database

```
ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES (property_name=property_value, ...);    -- (Note: SCHEMA added

ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;    -- (Note: Hive 0.13.0 and later; SCHEMA a
```

The uses of SCHEMA and DATABASE are interchangeable – they mean the same thing. ALTER SCHEMA was added in Hive 0.14 ([HIVE-6601](#)).

No other metadata about a database can be changed.

Use Database

```
USE database_name;
USE DEFAULT;
```

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()` (as of [Hive 0.13.0](#)).

USE database_name was added in Hive 0.6 ([HIVE-675](#)).

Create/Drop/Truncate Table

- [Create Table](#)
 - [Row Format, Storage Format, and SerDe](#)
 - [Partitioned Tables](#)
 - [External Tables](#)
 - [Create Table As Select \(CTAS\)](#)
 - [Create Table Like](#)
 - [Bucketed Sorted Tables](#)
 - [Skewed Tables](#)
 - [Temporary Tables](#)
- [Drop Table](#)
- [Truncate Table](#)

Create Table

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name      -- (Note: TEMPORARY available in Hive 0.1
[(col_name data_type [COMMENT col_comment], ...)]
[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 lat
]
[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
| 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
```

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

- Table names and column names are case insensitive but SerDe and property names are case sensitive.
 - In Hive 0.12 and earlier, only alphanumeric and underscore characters are allowed in table and column names.
 - In Hive 0.13 and later, column names can contain any **Unicode** character (see **HIVE-6013**). Any column name that is specified within backticks (`) is treated literally. Within a backtick string, use double backticks (``) to represent a backtick character. Backtick quotation also enables the use of reserved keywords for table and column identifiers.
 - To revert to pre-0.13.0 behavior and restrict column names to alphanumeric and underscore characters, set the configuration property **hive.support.quoted.identifiers** to none. In this configuration, backticked names are interpreted as regular expressions. For details, see **Supporting Quoted Identifiers in Column Names**.
- Table and column comments are string literals (single-quoted).
- The TBLPROPERTIES clause allows you to tag the table definition with your own metadata key/value pairs. Some predefined table properties also exist, such as last_modified_user and last_modified_time which are automatically added and managed by Hive. Other predefined table properties include:
 - TBLPROPERTIES ("comment"="*table_comment*")
 - TBLPROPERTIES ("hbase.table.name"="*table_name*") – see **HBase Integration**.
 - TBLPROPERTIES ("immutable"="true") or ("immutable"="false") in release 0.13.0+ (**HIVE-6406**) – see **Inserting Data into Hive Tables from Queries**.
 - TBLPROPERTIES ("orc.compress"="ZLIB") or ("orc.compress"="SNAPPY") or ("orc.compress"="NONE") and other ORC properties – see **ORC Files**.
 - TBLPROPERTIES ("transactional"="true") or ("transactional"="false") in release 0.14.0+, the default is "false" – see **Hive Transactions**.
 - TBLPROPERTIES ("NO_AUTO_COMPACTION"="true") or ("NO_AUTO_COMPACTION"="false"), the default is "false" – see **Hive Transactions**.
 - TBLPROPERTIES ("auto.purge"="true") or ("auto.purge"="false") in release 1.2.0+ (**HIVE-9118**) – see **Drop Table** and **Drop Partitions**.
- To specify a database for the table, either issue the **USE database_name** statement prior to the CREATE TABLE statement (in **Hive 0.6** and later) or qualify the table name with a database name ("database_name.table.name" in **Hive 0.7** and later). The keyword "default" can be used for the default database.

See **Alter Table** below for more information about table comments, table properties, and SerDe properties.

See **Type System** and **Hive Data Types** for details about the primitive and complex data types.

Row Format, Storage Format, and SerDe

You can create tables with a custom SerDe or using a native SerDe. A native SerDe is used if ROW FORMAT is not specified or ROW FORMAT DELIMITED is specified. You can use the DELIMITED clause to read delimited files, and you can enable escaping for the delimiter characters by using the 'ESCAPED BY' clause (such as ESCAPED BY '\') – escaping is needed if you want to work with data that can contain these delimiter characters. A custom NULL format can also be specified using the 'NULL DEFINED AS' clause (default is '\N'). Use the SERDE clause to create a table with a custom SerDe. For more information on SerDes see:

- **Hive SerDe**
- **SerDe**
- **HCatalog Storage Formats**

You must specify a list of columns for tables that use a native SerDe. Refer to the **Types** part of the User Guide for the allowable column types. A list of columns for tables that use a custom SerDe may be specified but Hive will query the SerDe to determine the actual list of columns for this table.

Use STORED AS TEXTFILE if the data needs to be stored as plain text files. TEXTFILE is the default file format, unless the configuration parameter **hive.default.fileformat** has a different setting.

Use STORED AS SEQUENCEFILE if the data needs to be compressed. Please read more about **CompressedStorage** if you are planning to keep data compressed in your Hive tables.

Use STORED AS ORC if the data needs to be stored in **ORC file format**.

Use ROW FORMAT SERDE for the RegEx SerDe, as shown in the example **Apache Weblog Data** in Getting Started.

Use INPUTFORMAT and OUTPUTFORMAT in the file_format to specify the name of a corresponding InputFormat and OutputFormat class as a string literal, for example, 'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextInputFormat'. For LZO compression, the values to use are 'INPUTFORMAT "com.hadoop.mapred.DeprecatedLzoTextInputFormat" OUTPUTFORMAT "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat"' (see **LZO Compression**).

Use STORED AS PARQUET (without ROW FORMAT SERDE) for the **Parquet** columnar storage format in **Hive 0.13.0 and later**; or use ROW FORMAT SERDE ... STORED AS INPUTFORMAT ... OUTPUTFORMAT ... in **Hive 0.10, 0.11, or 0.12**.

Use STORED AS AVRO for Avro files in **Hive 0.14.0 and later** (see **Avro SerDe**).

Use STORED BY to create a non-native table, for example in **HBase**. See **StorageHandlers** for more information on this option.

To change a table's SerDe or SERDEPROPERTIES, use the ALTER TABLE statement as described below in **Add SerDe Properties**.

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, you get this error: "FAILED: Error in semantic analysis: Column repeated in partitioning columns," it means you are trying to include the partitioned column in the data of the table itself. You probably really do have the column defined. However, the partition you create makes a pseudocolumn on which you can query, so you must rename your table column to something else (that users should not query on!).

For example, suppose your original unpartitioned table had three columns: id, date, and name.

Example:

```
id      int,
date    date,
name    varchar
```

Now you want to partition on date. Your Hive definition could use "dtDontQuery" as a column name so that "date" can be used for partitioning (and querying).

Example:

```
create table table_name (
  id          int,
  dtDontQuery string,
  name        string
)
partitioned by (date string)
```

Now your users will still query on "where date = '...'" but the second column dtDontQuery will hold the original values.

Here's an example statement to create a partitioned table:

Example:

```
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)
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.

Example:

```
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 above statement lets you create the same table as the previous table.

In the previous examples the data is stored in <hive.metastore.warehouse.dir>/page_view. Specify a value for the key [hive.metastore.warehouse.dir](#) in the Hive config file hive-site.xml.

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](#).

Example:

```
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>';
```

You can use the above statement to create a page_view table which points to any HDFS location for its storage. But you still have to make sure that the data is delimited as specified in the CREATE statement above.

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 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](#), the SELECT statement can include one or more common table expressions (CTEs), as shown in the [SELECT syntax](#). For an example, see [Common Table Expression](#).

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.

```
CREATE TABLE empty_key_value_store
LIKE key_value_store;
```

Before Hive 0.8.0, CREATE TABLE LIKE view_name would make a copy of the view. In Hive 0.8.0 and later releases, CREATE TABLE LIKE view_name creates a table by adopting the schema of view_name (fields and partition columns) using defaults for SerDe and file formats.

Bucketed Sorted Tables

Example:

```
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)
CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\001'
  COLLECTION ITEMS TERMINATED BY '\002'
  MAP KEYS TERMINATED BY '\003'
STORED AS SEQUENCEFILE;
```

In the example above, the page_view 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 sorting property allows internal operators to take advantage of the better-known data structure while evaluating queries, also increasing efficiency. MAP KEYS and COLLECTION ITEMS keywords can be used if any of the columns are lists or maps.

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.

There is also an example of [creating and populating bucketed tables](#).

Skewed Tables

Version information

As of Hive 0.10.0 ([HIVE-3072](#) and [HIVE-3649](#)). See [HIVE-3026](#) for additional JIRA tickets that implemented list bucketing in Hive 0.10.0 and 0.11.0.

Design documents

Read the [Skewed Join Optimization](#) and [List Bucketing](#) design documents for more information.

This feature can be used to improve performance for tables where one or more columns have [skewed](#) values. By specifying the values that appear very often (heavy skew) Hive will split those out into separate files (or directories in case of [list bucketing](#)) automatically and take this fact into account during queries so that it can skip or include the whole file (or directory in case of [list bucketing](#)) if possible.

This can be specified on a per-table level during table creation.

The following example shows one column with three skewed values, optionally with the STORED AS DIRECTORIES clause which specifies list bucketing.

Example:

```
CREATE TABLE list_bucket_single (key STRING, value STRING)
```

```
SKEWED BY (key) ON (1,5,6) [STORED AS DIRECTORIES];
```

And here is an example of a table with two skewed columns.

Example:

```
CREATE TABLE list_bucket_multiple (col1 STRING, col2 int, col3 STRING)
  SKEWED BY (col1, col2) ON (('s1',1), ('s3',3), ('s13',13), ('s78',78)) [STORED AS DIRECTORIES];
```

For corresponding ALTER TABLE statements, see [Alter Table Skewed or Stored as Directories](#) below.

Temporary Tables

 **Version information**
As of Hive 0.14.0 ([HIVE-7090](#)).

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.

Starting in [Hive 1.1.0](#) the storage policy for temporary tables can be set to memory, ssd, or default with the [hive.exec.temporary.table.storage](#) configuration parameter (see [HDFS Storage Types and Storage Policies](#)).

Drop Table


```
DROP TABLE [IF EXISTS] table_name [PURGE];      -- (Note: PURGE available in Hive 0.14.0 and later)
```

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).

Otherwise, the table information is removed from the metastore and the raw data is removed as if by 'hadoop dfs -rm'. In many cases, this results in the table data being moved into the user's .Trash folder in their home directory; users who mistakenly DROP TABLEs may thus be able to recover their lost data by recreating a table with the same schema, recreating any necessary partitions, and then moving the data back into place manually using Hadoop. This solution is subject to change over time or across installations as it relies on the underlying implementation; users are strongly encouraged not to drop tables capriciously.

 **Version information: PURGE**
The PURGE option is added in version 0.14.0 by [HIVE-7100](#).

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 (see [Create Table](#) above).

In Hive 0.7.0 or later, DROP returns an error if the table doesn't exist, unless IF EXISTS is specified or the configuration variable [hive.exec.drop.ignorenonexistent](#) is set to true.

See the Alter Partition section below for how to [drop partitions](#).

Truncate Table

 **Version information**
As of Hive 0.11.0 ([HIVE-446](#)).

```
TRUNCATE TABLE table_name [PARTITION partition_spec];

partition_spec:
    : (partition_column = partition_col_value, partition_column = partition_col_value, ...)
```

Removes all rows from a table or partition(s). Currently target table should be native/managed table or 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.

Alter Table/Partition/Column

- [Alter Table](#)
 - [Rename Table](#)
 - [Alter Table Properties](#)
 - [Alter Table Comment](#)
 - [Add SerDe Properties](#)
 - [Alter Table Storage Properties](#)
 - [Alter Table Skewed or Stored as Directories](#)
 - [Alter Table Skewed](#)
 - [Alter Table Not Skewed](#)

- [Alter Table Not Stored as Directories](#)
 - [Alter Table Set Skewed Location](#)
- [Additional Alter Table Statements](#)
- [Alter Partition](#)
 - [Add Partitions](#)
 - [Rename Partition](#)
 - [Exchange Partition](#)
 - [Recover Partitions \(MSCK REPAIR TABLE\)](#)
 - [Drop Partitions](#)
 - [\(Un\)Archive Partition](#)
- [Alter Either Table or Partition](#)
 - [Alter Table/Partition File Format](#)
 - [Alter Table/Partition Location](#)
 - [Alter Table/Partition Touch](#)
 - [Alter Table/Partition Protections](#)
 - [Alter Table/Partition Compact](#)
 - [Alter Table/Partition Concatenate](#)
- [Alter Column](#)
 - [Rules for Column Names](#)
 - [Change Column Name/Type/Position/Comment](#)
 - [Add/Replace Columns](#)
 - [Partial Partition Specification](#)

Alter table statements enable you to change the structure of an existing table. You can add columns/partitions, change SerDe, add table and SerDe properties, or rename the table itself. Similarly, alter table partition statements allow you change the properties of a specific partition in the named table.

Alter Table

Rename Table

```
ALTER TABLE table_name RENAME TO new_table_name;
```

This statement lets you change the name of a table to a different name.

As of version 0.6, a rename on a managed table moves its HDFS location as well. (Older Hive versions 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 [PARTITION partition_spec] SET SERDEPROPERTIES serde_properties;

serde_properties:
    : (property_name = property_value, property_name = property_value, ... )
```

These statements enable you to change a table's SerDe or add user-defined metadata to the table's SerDe object.

The SerDe properties are passed to the table's SerDe when it is being initialized by Hive to serialize and deserialize data. So users can store any information required for their custom SerDe here. Refer to the [SerDe documentation](#) and [Hive SerDe](#) in the Developer Guide for more information, and see [Row Format, Storage Format, and SerDe](#) above for details about setting a table's SerDe and SERDEPROPERTIES in a CREATE TABLE statement.

Note that both property_name and property_value must be quoted.

Example:

```
ALTER TABLE table_name SET SERDEPROPERTIES ('field.delim' = ',');
```

Alter Table Storage Properties

```
ALTER TABLE table_name CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name, ...)]
    INTO num_buckets BUCKETS;
```


These statements change the table's physical storage properties.

NOTE: These commands will only modify Hive's metadata, and will *NOT* reorganize or reformat existing data. Users should make sure the actual data layout conforms with the metadata definition.

Alter Table Skewed or Stored as Directories

**Version information**
As of Hive 0.10.0 ([HIVE-3072](#) and [HIVE-3649](#)). See [HIVE-3026](#) for additional JIRA tickets that implemented list bucketing in Hive 0.10.0 and 0.11.0.

A table's SKEWED and STORED AS DIRECTORIES options can be changed with ALTER TABLE statements. See [Skewed Tables](#) above for the corresponding CREATE TABLE syntax.

Alter Table Skewed

```
ALTER TABLE table_name SKEWED BY (col_name1, col_name2, ...)
  ON ([col_name1_value, col_name2_value, ...] [, (col_name1_value, col_name2_value), ...]
  [STORED AS DIRECTORIES];
```

The STORED AS DIRECTORIES option determines whether a [skewed](#) table uses the [list bucketing](#) 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.

Additional Alter Table Statements

See [Alter Either Table or Partition](#) below for more DDL statements that alter tables.

Alter Partition

Partitions can be added, renamed, exchanged (moved), dropped, or (un)archived by using the PARTITION clause in an ALTER TABLE statement, as described below. To make the metastore aware of partitions that were added directly to HDFS, you can use the metastore check command ([MSCK](#)) or on Amazon EMR you can use the RECOVER PARTITIONS option of ALTER TABLE. See [Alter Either Table or Partition](#) below for more ways to alter partitions.

**Version 1.2+**
As of Hive 1.2 ([HIVE-10307](#)), the partition values specified in partition specification are type checked, converted, and normalized to conform to their column types if the property [hive.typecheck.on.insert](#) is set to true (default). The values can be number literals.

Add Partitions

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec
  [LOCATION 'location1'] partition_spec [LOCATION 'location2'] ...;

partition_spec:
  : (partition_column = partition_col_value, partition_column = partition_col_value, ...)
```

You can use ALTER TABLE ADD PARTITION to add partitions to a table. Partition values should be quoted only if they are strings. The location must be a directory inside of which data files reside. (ADD PARTITION changes the table metadata, but does not load data. If the data does not exist in the partition's location, queries will not return any results.) An error is thrown if the partition_spec for the table already exists. You can use IF NOT EXISTS to skip the error.

**Version 0.7**
Although it is proper syntax to have multiple partition_spec in a single ALTER TABLE, if you do this in version 0.7 your partitioning scheme will fail. That is, every query specifying a partition will always use only the first partition.

Specifically, the following example will FAIL silently and without error in Hive 0.7, and all queries will go only to dt='2008-08-08' partition, no matter which partition you specify.

Example:

```
ALTER TABLE page_view ADD PARTITION (dt='2008-08-08', country='us') location '/path/to/us/part080808'
                                PARTITION (dt='2008-08-09', country='us') location '/path/to/us/part080809';
```

In Hive 0.8 and later, you can add multiple partitions in a single ALTER TABLE statement as shown in the previous example.

In Hive 0.7, if you want to add many partitions you should use the following form:

```
ALTER TABLE table_name ADD PARTITION (partCol = 'value1') location 'loc1';
ALTER TABLE table_name ADD PARTITION (partCol = 'value2') location 'loc2';
...
ALTER TABLE table_name ADD PARTITION (partCol = 'valueN') location 'locN';
```

Dynamic Partitions

Partitions can be added to a table dynamically, using a Hive INSERT statement (or a Pig STORE statement). See these documents for details and examples:

- [Design Document for Dynamic Partitions](#)
- [Tutorial: Dynamic-Partition Insert](#)
- [Hive DML: Dynamic Partition Inserts](#)
- [HCatalog Dynamic Partitioning](#)
 - [Usage with Pig](#)
 - [Usage from MapReduce](#)

Rename Partition

 **Version information**
As of Hive 0.9.

```
ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION partition_spec;
```

This statement lets you change the value of a partition column. One of use cases is that you can use this statement to normalize your legacy partition column value to conform to its type. In this case, the type conversion and normalization are not enabled for the column values in old *partition_spec* even with property [hive.typecheck.on.insert](#) set to true (default) which allows you to specify any legacy data in form of string in the old *partition_spec*.

Exchange Partition

Partitions can be exchanged (moved) between tables.

 **Version information**
As of Hive 0.12 ([HIVE-4095](#)). Multiple partitions supported in Hive versions [1.2.2](#), [1.3.0](#), and [2.0.0+](#).

```
ALTER TABLE table_name_1 EXCHANGE PARTITION (partition_spec) WITH TABLE table_name_2;
-- multiple partitions
ALTER TABLE table_name_1 EXCHANGE PARTITION (partition_spec, partition_spec2, ...) WITH TABLE table_name_2;
```

This statement lets you move the data in a partition from a table to another table that has the same schema and does not already have that partition. For further details on this feature, see [Exchange Partition](#) and [HIVE-4095](#).

Recover Partitions (MSCK REPAIR TABLE)

Hive stores a list of partitions for each table in its metastore. If, however, new partitions are directly added to HDFS (say by using `hadoop fs -put` command), the metastore (and hence Hive) will not be aware of these partitions unless the user runs `ALTER TABLE table_name ADD PARTITION` commands on each of the newly added partitions.

However, users can run a metastore check command with the repair table option:

```
MSCK REPAIR TABLE table_name;
```

which will add metadata about partitions to the Hive metastore for partitions for which such metadata doesn't already exist. In other words, it will add any partitions that exist on HDFS but not in metastore to the metastore. See [HIVE-874](#) for more details.

The equivalent command on Amazon Elastic MapReduce (EMR)'s version of Hive is:

```
ALTER TABLE table_name RECOVER PARTITIONS;
```

Starting with Hive 1.3, MSCK will throw exceptions if directories with disallowed characters in partition values are found on HDFS. Use **hive.msck.path.validation** setting on the client to alter this behavior; "skip" will simply skip the directories. "ignore" will try to create partitions anyway (old behavior). This may or may not work.

Drop Partitions

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec[, PARTITION partition_spec, ...]
[IGNORE PROTECTION] [PURGE];           -- (Note: PURGE available in Hive 1.2.0 and later, IGNORE PROTECTION not av
```

You can use `ALTER TABLE DROP PARTITION` to drop a partition for a table. This removes the data and metadata for this partition. The data is actually moved to the `.Trash/Current` directory if Trash is configured, unless `PURGE` is specified, but the metadata is completely lost (see [Drop Table](#) above).

Version Information: PROTECTION
IGNORE PROTECTION is no longer available in versions 2.0.0 and later. This functionality is replaced by using one of the several security options available with Hive (see [SQL Standard Based Hive Authorization](#)). See [HIVE-11145](#) for details.

For tables that are protected by [NO_DROP CASCADE](#), you can use the predicate IGNORE PROTECTION to drop a specified partition or set of partitions (for example, when splitting a table between two Hadoop clusters):

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec IGNORE PROTECTION;
```

The above command will drop that partition regardless of protection stats.

Version information: PURGE
The PURGE option is added to ALTER TABLE in version 1.2.1 by [HIVE-10934](#).

If PURGE is specified, the partition data does not go to the .Trash/Current directory and so cannot be retrieved in the event of a mistaken DROP:

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec PURGE;      -- (Note: Hive 1.2.0 and later)
```

The purge option can also be specified with the table property auto.purge (see [Create Table](#) above).

In Hive 0.7.0 or later, DROP returns an error if the partition doesn't exist, unless IF EXISTS is specified or the configuration variable [hive.exec.drop.ignorenonexistent](#) is set to true.

```
ALTER TABLE page_view DROP PARTITION (dt='2008-08-08', country='us');
```

(Un)Archive Partition

```
ALTER TABLE table_name ARCHIVE PARTITION partition_spec;
ALTER TABLE table_name UNARCHIVE PARTITION partition_spec;
```

Archiving is a feature to moves a partition's files into a Hadoop Archive (HAR). Note that only the file count will be reduced; HAR does not provide any compression. See [LanguageManual Archiving](#) for more information

Alter Either Table or Partition

Alter Table/Partition File Format

```
ALTER TABLE table_name [PARTITION partition_spec] SET FILEFORMAT file_format;
```

This statement changes the table's (or partition's) file format. For available file_format options, see the section above on [CREATE TABLE](#).

Alter Table/Partition Location

```
ALTER TABLE table_name [PARTITION partition_spec] SET LOCATION "new location";
```

Alter Table/Partition Touch

```
ALTER TABLE table_name TOUCH [PARTITION partition_spec];
```

TOUCH reads the metadata, and writes it back. This has the effect of causing the pre/post execute hooks to fire. An example use case is if you have a hook that logs all the tables/partitions that were modified, along with an external script that alters the files on HDFS directly. Since the script modifies files outside of hive, the modification wouldn't be logged by the hook. The external script could call TOUCH to fire the hook and mark the said table or partition as modified.

Also, it may be useful later if we incorporate reliable last modified times. Then touch would update that time as well.

Note that TOUCH doesn't create a table or partition if it doesn't already exist. (See [Create Table](#).)

Alter Table/Partition Protections

Version information
As of Hive 0.7.0 ([HIVE-1413](#)). The CASCADE clause for NO_DROP was added in HIVE 0.8.0 ([HIVE-2605](#)).

This functionality was removed in Hive 2.0.0. This functionality is replaced by using one of the several security options available with Hive (see [SQL Standard Based Hive Authorization](#)). See [HIVE-11145](#) for details.

```
ALTER TABLE table_name [PARTITION partition_spec] ENABLE|DISABLE NO_DROP [CASCADE];

ALTER TABLE table_name [PARTITION partition_spec] ENABLE|DISABLE OFFLINE;
```

Protection on data can be set at either the table or partition level. Enabling NO_DROP prevents a table from being [dropped](#). Enabling OFFLINE prevents the data in a table or partition from being queried, but the metadata can still be accessed.

If any partition in a table has NO_DROP enabled, the table cannot be dropped either. Conversely, if a table has NO_DROP enabled then partitions may be

dropped, but with NO_DROP CASCADE partitions cannot be dropped either unless the [drop partition command](#) specifies IGNORE PROTECTION.

Alter Table/Partition Compact

Version information
In Hive release [0.13.0](#) and later when [transactions](#) are being used, the ALTER TABLE statement can request [compaction](#) of a table or partition.

```
ALTER TABLE table_name [PARTITION (partition_key = 'partition_value' [, ...])]
  COMPACT 'compaction_type';
```

In general you do not need to request compactions when [Hive transactions](#) are being used, because the system will detect the need for them and initiate the compaction. However, if compaction is turned off for a table or you want to compact the table at a time the system would not choose to, ALTER TABLE can initiate the compaction. The statement will enqueue a request for compaction and return. To watch the progress of the compaction, use [SHOW COMPACTIONS](#).

The compaction_type can be MAJOR or MINOR. See the Basic Design section in [Hive Transactions](#) for more information.

Alter Table/Partition Concatenate

Version information
In Hive release [0.8.0](#) RCFile added support for fast block level merging of small RCFiles using concatenate command. In Hive release [0.14.0](#) ORC files added support fast stripe level merging of small ORC files using concatenate command.

```
ALTER TABLE table_name [PARTITION (partition_key = 'partition_value' [, ...])] CONCATENATE;
```

If the table or partition contains many small RCFiles or ORC files, then the above command will merge them into larger files. In case of RCFile the merge happens at block level whereas for ORC files the merge happens at stripe level thereby avoiding the overhead of decompressing and decoding the data.

Alter Column

Rules for Column Names

Column names are case insensitive.

Version information
In Hive release 0.12.0 and earlier, column names can only contain alphanumeric and underscore characters.

In Hive release 0.13.0 and later, by default column names can be specified within backticks (`) and contain any [Unicode](#) character ([HIVE-6013](#)). Within a string delimited by backticks, all characters are treated literally except that double backticks (``) represent one backtick character. The pre-0.13.0 behavior can be used by setting [hive.support.quoted.identifiers](#) to none, in which case backticked names are interpreted as regular expressions. See [Supporting Quoted Identifiers in Column Names](#) for details.

Backtick quotation enables the use of reserved keywords for column names, as well as table names.

Change Column Name/Type/Position/Comment

```
ALTER TABLE table_name [PARTITION partition_spec] CHANGE [COLUMN] col_old_name col_new_name column_type
  [COMMENT col_comment] [FIRST|AFTER column_name] [CASCADE|RESTRICT];
```

This command will allow users to change a column's name, [data type](#), comment, or position, or an arbitrary combination of them. The PARTITION clause is available in [Hive 0.14.0](#) and later; see [Upgrading Pre-Hive 0.13.0 Decimal Columns](#) for usage. A patch for Hive 0.13 is also available (see [HIVE-7971](#)).

The CASCADE|RESTRICT clause is available in [Hive 0.15.0](#). ALTER TABLE CHANGE COLUMN with CASCADE command changes the columns of a table's metadata, and cascades the same change to all the partition metadata. RESTRICT is the default, limiting column change only to table metadata.

Warning: ALTER TABLE CHANGE COLUMN CASCADE clause will override the table partition's column metadata regardless of the table or partition's [protection mode](#). Use with discretion.

Warning: The column change command will only modify Hive's metadata, and will not modify data. Users should make sure the actual data layout of the table/partition conforms with the metadata definition.

Example:

```
CREATE TABLE test_change (a int, b int, c int);

// First change column a's name to a1.
ALTER TABLE test_change CHANGE a a1 INT;

// Next change column a1's name to a2, its data type to string, and put it after column b.
ALTER TABLE test_change CHANGE a1 a2 STRING AFTER b;
// The new table's structure is:  b int, a2 string, c int.

// Then change column c's name to c1, and put it as the first column.
ALTER TABLE test_change CHANGE c c1 INT FIRST;
// The new table's structure is:  c1 int, b int, a2 string.
```



```
// Add a comment to column a1
ALTER TABLE test_change CHANGE a1 a1 INT COMMENT 'this is column a1';
```

Add/Replace Columns


```
ALTER TABLE table_name
  [PARTITION partition_spec]           -- (Note: Hive 0.14.0 and later)
ADD|REPLACE COLUMNS (col_name data_type [COMMENT col_comment], ...)
  [CASCADE|RESTRICT]                   -- (Note: Hive 0.15.0 and later)
```


ADD COLUMNS lets you add new columns to the end of the existing columns but before the partition columns. This is supported for Avro backed tables as well, for [Hive 0.14](#) and later.

REPLACE COLUMNS removes all existing columns and adds the new set of columns. This can be done only for tables with a native SerDe (DynamicSerDe, MetadataTypedColumnsetSerDe, LazySimpleSerDe and ColumnarSerDe). Refer to [Hive SerDe](#) for more information. REPLACE COLUMNS can also be used to drop columns. For example, "ALTER TABLE test_change REPLACE COLUMNS (a int, b int);" will remove column 'c' from test_change's schema.

The PARTITION clause is available in [Hive 0.14.0](#) and later; see [Upgrading Pre-Hive 0.13.0 Decimal Columns](#) for usage.

The CASCADE|RESTRICT clause is available in [Hive 0.15.0](#). ALTER TABLE ADD|REPLACE COLUMNS with CASCADE command changes the columns of a table's metadata, and cascades the same change to all the partition metadata. RESTRICT is the default, limiting column changes only to table metadata.

 ALTER TABLE ADD or REPLACE COLUMNS CASCADE will override the table partition's column metadata regardless of the table or partition's [protection mode](#). Use with discretion.

 The column change command will only modify Hive's metadata, and will not modify data. Users should make sure the actual data layout of the table/partition conforms with the metadata definition.

Partial Partition Specification

As of Hive 0.14 ([HIVE-8411](#)), users are able to provide a partial partition spec for certain above alter column statements, similar to dynamic partitioning. So rather than having to issue an alter column statement for each partition that needs to be changed:

```
ALTER TABLE foo PARTITION (ds='2008-04-08', hr=11) CHANGE COLUMN dec_column_name dec_column_name DECIMAL(38,18);
ALTER TABLE foo PARTITION (ds='2008-04-08', hr=12) CHANGE COLUMN dec_column_name dec_column_name DECIMAL(38,18);
...
```

... you can change many existing partitions at once using a single ALTER statement with a partial partition specification:

```
// hive.exec.dynamic.partition needs to be set to true to enable dynamic partitioning with ALTER PARTITION
SET hive.exec.dynamic.partition = true;

// This will alter all existing partitions in the table with ds='2008-04-08' -- be sure you know what you are doing!
ALTER TABLE foo PARTITION (ds='2008-04-08', hr) CHANGE COLUMN dec_column_name dec_column_name DECIMAL(38,18);


// This will alter all existing partitions in the table -- be sure you know what you are doing!
ALTER TABLE foo PARTITION (ds, hr) CHANGE COLUMN dec_column_name dec_column_name DECIMAL(38,18);
```

Similar to dynamic partitioning, [hive.exec.dynamic.partition](#) must be set to true to enable use of partial partition specs during ALTER PARTITION. This is supported for the following operations:

- Change column
- Add column
- Replace column
- File Format
- Serde Properties

Create/Drop/Alter View

- [Create View](#)
- [Drop View](#)
- [Alter View Properties](#)
- [Alter View As Select](#)

 **Version information**
View support is only available in Hive 0.6 and later.

Create View

```
CREATE VIEW [IF NOT EXISTS] [db_name.]view_name [(column_name [COMMENT column_comment], ...) ]
  [COMMENT view_comment]
```

```
[TBLPROPERTIES (property_name = property_value, ...)]
AS SELECT ...;
```

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

If no column names are supplied, the names of the view's columns will be derived automatically from the defining SELECT expression. (If the SELECT contains unaliased scalar expressions such as x+y, the resulting view column names will be generated in the form _C0, _C1, etc.) When renaming columns, column comments can also optionally be supplied. (Comments are not automatically inherited from underlying columns.)

A CREATE VIEW statement will fail if the view's defining SELECT expression is invalid.

Note that a view is a purely logical object with no associated storage. (No support for materialized views is currently available in Hive.) When a query references a view, the view's definition is evaluated in order to produce a set of rows for further processing by the query. (This is a conceptual description; in fact, as part of query optimization, Hive may combine the view's definition with the query's, e.g. pushing filters from the query down into the view.)

A view's schema is frozen at the time the view is created; subsequent changes to underlying tables (e.g. adding a column) will not be reflected in the view's schema. If an underlying table is dropped or changed in an incompatible fashion, subsequent attempts to query the invalid view will fail.

Views are read-only and may not be used as the target of LOAD/INSERT/ALTER. For changing metadata, see [ALTER VIEW](#).

A view may contain ORDER BY and LIMIT clauses. If a referencing query also contains these clauses, the query-level clauses are evaluated **after** the view clauses (and after any other operations in the query). For example, if a view specifies LIMIT 5, and a referencing query is executed as (select * from v LIMIT 10), then at most 5 rows will be returned.

Starting with [Hive 0.13.0](#), the view's select statement can include one or more common table expressions (CTEs) as shown in the [SELECT syntax](#). For examples of CTEs in CREATE VIEW statements, see [Common Table Expression](#).

Example:

```
CREATE VIEW onion_referrers(url COMMENT 'URL of Referring page')
  COMMENT 'Referrers to The Onion website'
AS
SELECT DISTINCT referrer_url
FROM page_view
WHERE page_url='http://www.theonion.com';
```

Use [SHOW CREATE TABLE](#) to display the CREATE VIEW statement that created a view.

Drop View

```
DROP VIEW [IF EXISTS] [db_name.]view_name;
```

DROP VIEW removes metadata for the specified view. (It is illegal to use DROP TABLE on a view.)

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

In Hive 0.7.0 or later, DROP returns an error if the view doesn't exist, unless IF EXISTS is specified or the configuration variable [hive.exec.drop.ignorenonexistent](#) is set to true.

Example:

```
DROP VIEW onion_referrers;
```

Alter View Properties


```
ALTER VIEW [db_name.]view_name SET TBLPROPERTIES table_properties;
```

table_properties:

```
: (property_name = property_value, property_name = property_value, ...)
```

As with ALTER TABLE, you can use this statement to add your own metadata to a view.

Alter View As Select

 **Version information**
As of [Hive 0.11](#).

```
ALTER VIEW [db_name.]view_name AS select_statement;
```

Alter View As Select changes the definition of a view, which must exist. The syntax is similar to that for CREATE VIEW and the effect is the same as for CREATE OR REPLACE VIEW.

Note: The view must already exist, and if the view has partitions, it could not be replaced by Alter View As Select.

Create/Drop/Alter Index

Version information

This section provides a brief introduction to Hive indexes, which are documented more fully here:

- [Overview of Hive Indexes](#)
- [Indexes design document](#)

In Hive 0.12.0 and earlier releases, the index name is case-sensitive for CREATE INDEX and DROP INDEX statements. However, ALTER INDEX requires an index name that was created with lowercase letters (see [HIVE-2752](#)). This bug is fixed in [Hive 0.13.0](#) by making index names case-insensitive for all HiveQL statements. For releases prior to 0.13.0, the best practice is to use lowercase letters for all index names.

Create Index

```
CREATE INDEX index_name
  ON TABLE base_table_name (col_name, ...)
  AS index_type
  [WITH DEFERRED REBUILD]
  [IDXPROPERTIES (property_name=property_value, ...)]
  [IN TABLE index_table_name]
  [
    [ ROW FORMAT ...] STORED AS ...
    | STORED BY ...
  ]
  [LOCATION hdfs_path]
  [TBLPROPERTIES (...)]
  [COMMENT "index comment"];
```

CREATE INDEX creates an index on a table using the given list of columns as keys. See CREATE INDEX in the [Indexes](#) design document.

Drop Index

```
DROP INDEX [IF EXISTS] index_name ON table_name;
```

DROP INDEX drops the index, as well as deleting the index table.

In Hive 0.7.0 or later, DROP returns an error if the index doesn't exist, unless IF EXISTS is specified or the configuration variable [hive.exec.drop.ignorenonexistent](#) is set to true.

Alter Index

```
ALTER INDEX index_name ON table_name [PARTITION partition_spec] REBUILD;
```

ALTER INDEX ... REBUILD builds an index that was created using the WITH DEFERRED REBUILD clause, or rebuilds a previously built index. If PARTITION is specified, only that partition is rebuilt.

Create/Drop Macro

 **Version information**
As of [Hive 0.12.0](#).

Bug fixes:

- Prior to [Hive 1.3.0 and 2.0.0](#) when a HiveQL macro was used more than once while processing the same row, Hive returned the same result for all invocations even though the arguments were different. (See [HIVE-11432](#).)
- Prior to [Hive 1.3.0 and 2.0.0](#) when multiple macros were used while processing the same row, an ORDER BY clause could give wrong results. (See [HIVE-12277](#).)
- Prior to [Hive 2.1.0](#) when multiple macros were used while processing the same row, results of the later macros were overwritten by that of the first. (See [HIVE-13372](#).)

Hive 0.12.0 introduced macros to HiveQL, prior to which they could only be created in Java.

Create Temporary Macro

```
CREATE TEMPORARY MACRO macro_name([col_name col_type, ...]) expression;
```

CREATE TEMPORARY MACRO creates a macro using the given optional list of columns as inputs to the expression. Macros exist for the duration of the current session.

Examples:

```
CREATE TEMPORARY MACRO fixed_number() 42;
CREATE TEMPORARY MACRO string_len_plus_two(x string) length(x) + 2;
CREATE TEMPORARY MACRO simple_add (x int, y int) x + y;
```

Drop Temporary Macro

```
DROP TEMPORARY MACRO [IF EXISTS] macro_name;
```

DROP TEMPORARY MACRO returns an error if the function doesn't exist, unless IF EXISTS is specified.

Create/Drop/Reload Function

Temporary Functions

Create Temporary Function

```
CREATE TEMPORARY FUNCTION function_name AS class_name;
```

This statement lets you create a function that is implemented by the class_name. You can use this function in Hive queries as long as the session lasts. You can use any class that is in the class path of Hive. You can add jars to class path by executing 'ADD JAR' statements. Please refer to the CLI section [Hive Interactive Shell Commands](#), including [Hive Resources](#), for more information on how to add/delete files from the Hive classpath. Using this, you can register User Defined Functions (UDF's).

Also see [Hive Plugins](#) for general information about creating custom UDFs.

Drop Temporary Function

You can unregister a UDF as follows:


```
DROP TEMPORARY FUNCTION [IF EXISTS] function_name;
```

In Hive 0.7.0 or later, DROP returns an error if the function doesn't exist, unless IF EXISTS is specified or the configuration variable [hive.exec.drop.ignorenonexistent](#) is set to true.

Permanent Functions

In Hive 0.13 or later, functions can be registered to the metastore, so they can be referenced in a query without having to create a temporary function each session.

Create Function

 **Version information**
As of Hive 0.13.0 ([HIVE-6047](#)).

```
CREATE FUNCTION [db_name.]function_name AS class_name
[USING JAR|FILE|ARCHIVE 'file_uri' [, JAR|FILE|ARCHIVE 'file_uri'] ];
```

This statement lets you create a function that is implemented by the class_name. Jars, files, or archives which need to be added to the environment can be specified with the USING clause; when the function is referenced for the first time by a Hive session, these resources will be added to the environment as if [ADD JAR/FILE](#) had been issued. If Hive is not in local mode, then the resource location must be a non-local URI such as an HDFS location.

The function will be added to the database specified, or to the current database at the time that the function was created. The function can be referenced by fully qualifying the function name (db_name.function_name), or can be referenced without qualification if the function is in the current database.

Drop Function

 **Version information**
As of Hive 0.13.0 ([HIVE-6047](#)).

```
DROP FUNCTION [IF EXISTS] function_name;
```

DROP returns an error if the function doesn't exist, unless IF EXISTS is specified or the configuration variable [hive.exec.drop.ignorenonexistent](#) is set to true.

Reload Function

 **Version information**
As of Hive 1.2.0 ([HIVE-2573](#)).

```
RELOAD FUNCTION;
```

As of [HIVE-2573](#), creating permanent functions in one Hive CLI session may not be reflected in HiveServer2 or other Hive CLI sessions, if they were started before the function was created. Issuing RELOAD FUNCTION within a HiveServer2 or HiveCLI session will allow it to pick up any changes to the permanent functions that may have been done by a different HiveCLI session.

Create/Drop/Grant/Revoke Roles and Privileges

[Hive Default Authorization - Legacy Mode](#) has information about these DDL statements:

- [CREATE ROLE](#)
- [GRANT ROLE](#)
- [REVOKE ROLE](#)
- [GRANT privilege_type](#)
- [REVOKE privilege_type](#)
- [DROP ROLE](#)
- [SHOW ROLE GRANT](#)
- [SHOW GRANT](#)

For [SQL standard based authorization](#) in Hive 0.13.0 and later releases, see these DDL statements:

- Role Management Commands
 - [CREATE ROLE](#)
 - [GRANT ROLE](#)
 - [REVOKE ROLE](#)
 - [DROP ROLE](#)
 - [SHOW ROLES](#)
 - [SHOW ROLE GRANT](#)
 - [SHOW CURRENT ROLES](#)
 - [SET ROLE](#)
 - [SHOW PRINCIPALS](#)
- Object Privilege Commands
 - [GRANT privilege_type](#)
 - [REVOKE privilege_type](#)
 - [SHOW GRANT](#)

Show

- [Show Databases](#)
- [Show Tables/Partitions/Indexes](#)
 - [Show Tables](#)
 - [Show Partitions](#)
 - [Show Table/Partition Extended](#)
 - [Show Table Properties](#)
 - [Show Create Table](#)
 - [Show Indexes](#)
- [Show Columns](#)
- [Show Functions](#)
- [Show Granted Roles and Privileges](#)
- [Show Locks](#)
- [Show Conf](#)
- [Show Transactions](#)
- [Show Compactions](#)

These statements provide a way to query the Hive metastore for existing data and metadata accessible to this Hive system.

Show Databases

```
SHOW (DATABASES|SCHEMAS) [LIKE 'identifier_with_wildcards'];
```

SHOW DATABASES or SHOW SCHEMAS lists all of the databases defined in the metastore. The uses of SCHEMAS and DATABASES are interchangeable – they mean the same thing.

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'.

Show Tables/Partitions/Indexes

Show Tables


```
SHOW TABLES [IN database_name] ['identifier_with_wildcards'];
```

SHOW TABLES lists all the base tables and views in the current database (or the one explicitly named using the IN clause) with names matching the optional regular expression. Wildcards in the regular expression can only be '*' for any character(s) or '|' for a choice. Examples are 'page_view', 'page_v*', '*view|page*', all which will match the 'page_view' table. Matching tables are listed in alphabetical order. It is not an error if there are no matching tables found in metastore. If no regular expression is given then all tables in the selected database are listed.

Show Partitions

```
SHOW PARTITIONS table_name;
```

SHOW PARTITIONS lists all the existing partitions for a given base table. Partitions are listed in alphabetical order.

 **Version information**
As of Hive 0.6, SHOW PARTITIONS can filter the list of partitions as shown below.

It is also possible to specify parts of a partition specification to filter the resulting list.

Examples:

```
SHOW PARTITIONS table_name PARTITION(ds='2010-03-03');      -- (Note: Hive 0.6 and later)
SHOW PARTITIONS table_name PARTITION(hr='12');              -- (Note: Hive 0.6 and later)
SHOW PARTITIONS table_name PARTITION(ds='2010-03-03', hr='12');  -- (Note: Hive 0.6 and later)
```

 **Version information**
Starting with Hive 0.13.0, SHOW PARTITIONS can specify a database ([HIVE-5912](#)).

```
SHOW PARTITIONS [db_name.]table_name [PARTITION(partition_spec)];    -- (Note: Hive 0.13.0 and later)
```

Example:

```
SHOW PARTITIONS databaseFoo.tableBar PARTITION(ds='2010-03-03', hr='12');    -- (Note: Hive 0.13.0 and later)
```

Show Table/Partition Extended

```
SHOW TABLE EXTENDED [IN|FROM database_name] LIKE 'identifier_with_wildcards' [PARTITION(partition_spec)];
```

SHOW TABLE EXTENDED will list information for all tables matching the given regular expression. Users cannot use regular expression for table name if a partition specification is present. This command's output includes basic table information and file system information like totalNumberFiles, totalFileSize, maxFileSize, minFileSize,lastAccessTime, and lastUpdateTime. If partition is present, it will output the given partition's file system information instead of table's file system information.

Example

```
hive> show table extended like part_table;
OK
tableName:part_table
owner:thejas
location:file:/tmp/warehouse/part_table
inputformat:org.apache.hadoop.mapred.TextInputFormat
outputformat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
columns:struct columns { i32 i}
partitioned:true
partitionColumns:struct partition_columns { string d}
totalNumberFiles:1
totalFileSize:2
maxFileSize:2
minFileSize:2
lastAccessTime:0
lastUpdateTime:1459382233000
```

Show Table Properties

 **Version information**
As of Hive 0.10.0.

```
SHOW TBLPROPERTIES tblname;
SHOW TBLPROPERTIES tblname("foo");
```

The first form lists all of the table properties for the table in question one per row separated by tabs. The second form of the command prints only the value for the property that's being asked for.

Show Create Table

 **Version information**
As of [Hive 0.10](#).

```
SHOW CREATE TABLE ([db_name.]table_name|view_name);
```

SHOW CREATE TABLE shows the CREATE TABLE statement that creates a given table, or the CREATE VIEW statement that creates a given view.

Show Indexes

 **Version information**
As of Hive 0.7.

```
SHOW [FORMATTED] (INDEX|INDEXES) ON table_with_index [(FROM|IN) db_name];
```

SHOW INDEXES shows all of the indexes on a certain column, as well as information about them: index name, table name, names of the columns used as keys, index table name, index type, and comment. If the FORMATTED keyword is used, then column titles are printed for each column.

Show Columns

 **Version information**
As of [Hive 0.10](#).

```
SHOW COLUMNS (FROM|IN) table_name [(FROM|IN) db_name];
```

SHOW COLUMNS shows all the columns in a table including partition columns.

Show Functions

```
SHOW FUNCTIONS "a.*";
```

SHOW FUNCTIONS lists all the user defined and builtin functions matching the regular expression. To get all functions use "."

Show Granted Roles and Privileges

[Hive Default Authorization - Legacy Mode](#) has information about these SHOW statements:

- [SHOW ROLE GRANT](#)
- [SHOW GRANT](#)

In Hive 0.13.0 and later releases, [SQL standard based authorization](#) has these SHOW statements:

- [SHOW ROLE GRANT](#)
- [SHOW GRANT](#)
- [SHOW CURRENT ROLES](#)
- [SHOW ROLES](#)
- [SHOW PRINCIPALS](#)

Show Locks

```
SHOW LOCKS <table_name>;
SHOW LOCKS <table_name> EXTENDED;
SHOW LOCKS <table_name> PARTITION (<partition_spec>);
SHOW LOCKS <table_name> PARTITION (<partition_spec>) EXTENDED;
SHOW LOCKS (DATABASE|SCHEMA) database_name;      -- (Note: Hive 0.13.0 and later; SCHEMA added in Hive 0.14.0)
```

SHOW LOCKS displays the locks on a table or partition. See [Hive Concurrency Model](#) for information about locks.

SHOW LOCKS (DATABASE|SCHEMA) is supported from Hive 0.13 for DATABASE (see [HIVE-2093](#)) and Hive 0.14 for SCHEMA (see [HIVE-6601](#)). SCHEMA and DATABASE are interchangeable – they mean the same thing.

When [Hive transactions](#) are being used, SHOW LOCKS returns this information (see [HIVE-6460](#)):

- database name
- table name
- partition name (if the table is partitioned)
- the state the lock is in, which can be:
 - "acquired" – the requestor holds the lock
 - "waiting" – the requestor is waiting for the lock
 - "aborted" – the lock has timed out but has not yet been cleaned up
- Id of the lock blocking this one, if this lock is in "waiting" state
- the type of lock, which can be:
 - "exclusive" – no one else can hold the lock at the same time (obtained mostly by DDL operations such as drop table)
 - "shared_read" – any number of other shared_read locks can lock the same resource at the same time (obtained by reads; confusingly, an insert operation also obtains a shared_read lock)
 - "shared_write" – any number of shared_read locks can lock the same resource at the same time, but no other shared_write locks are allowed (obtained by update and delete)
- ID of the transaction this lock is associated with, if there is one
- last time the holder of this lock sent a heartbeat indicating it was still alive
- the time the lock was acquired, if it has been acquired
- Hive user who requested the lock
- host the user is running on

Show Conf

 **Version information**
As of [Hive 0.14.0](#).


```
SHOW CONF <configuration_name>;
```

SHOW CONF returns a description of the specified [configuration property](#).

- default value
- required type
- description

Note that SHOW CONF does not show the *current value* of a configuration property. For current property settings, use the "set" command in the CLI or a HiveQL script (see [Commands](#)) or in Beeline (see [Beeline Hive Commands](#)).

Show Transactions


 **Version information**
As of [Hive 0.13.0](#) (see [Hive Transactions](#)).

```
SHOW  TRANSACTIONS;
```

SHOW TRANSACTIONS is for use by administrators when [Hive transactions](#) are being used. It returns a list of all currently open and aborted transactions in the system, including this information:

- transaction ID
- transaction state
- user who started the transaction
- machine where the transaction was started

Show Compactions

 **Version information**
As of [Hive 0.13.0](#) (see [Hive Transactions](#)).

```
SHOW  COMPACTIONS;
```

SHOW COMPACTIONS returns a list of all tables and partitions currently being [compacted](#) or scheduled for compaction when [Hive transactions](#) are being used, including this information:

- database name
- table name
- partition name (if the table is partitioned)
- whether it is a major or minor compaction
- the state the compaction is in, which can be:
 - "initiated" – waiting in the queue to be compacted
 - "working" – being compacted
 - "ready for cleaning" – the compaction has been done and the old files are scheduled to be cleaned
- thread ID of the worker thread doing the compaction (only if in working state)
- the time at which the compaction started (only if in working or ready for cleaning state)

Compactions are initiated automatically, but can also be initiated manually with an [ALTER TABLE COMPACT statement](#).

Describe

- [Describe Database](#)
- [Describe Table/View/Column](#)
 - [Display Column Statistics](#)
- [Describe Partition](#)
- [Hive 2.0+: Syntax Change](#)

Describe Database

 **Version information**
As of Hive 0.7.

```
DESCRIBE DATABASE [EXTENDED] db_name;
DESCRIBE SCHEMA [EXTENDED] db_name;      -- (Note: Hive 0.15.0 and later)
```

DESCRIBE DATABASE shows the name of the database, its comment (if one has been set), and its root location on the filesystem. The uses of SCHEMA and DATABASE are interchangeable – they mean the same thing. DESCRIBE SCHEMA is added in Hive 0.15 ([HIVE-8803](#)).

EXTENDED also shows the [database properties](#).

Describe Table/View/Column

There are two formats for the describe table/view/column syntax, depending on whether or not the database is specified.

If the database is not specified, the optional column information is provided after a dot:

```
DESCRIBE [EXTENDED|FORMATTED]
  table_name[.col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )* ];
                                     -- (Note: Hive 1.x.x and 0.x.x only. See "Hive 2.0+: New Syntax" below)
```

If the database is specified, the optional column information is provided after a space:

```
DESCRIBE [EXTENDED|FORMATTED]
  [db_name.]table_name[ col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )* ];
                                     -- (Note: Hive 1.x.x and 0.x.x only. See "Hive 2.0+: New Syntax" below)
```

DESCRIBE shows the list of columns including partition columns for the given table. If the EXTENDED keyword is specified then it will show all the metadata for the table in Thrift serialized form. This is generally only useful for debugging and not for general use. If the FORMATTED keyword is specified,


```
Detailed Partition Information Partition(values:[abc], dbName:default, tableName:part_table, createTime:1459382234,
Time taken: 0.325 seconds, Fetched: 9 row(s)

hive> DESCRIBE formatted part_table partition (d='abc');
OK
# col_name          data_type          comment

i                   int


# Partition Information
# col_name          data_type          comment

d                   string

# Detailed Partition Information
Partition Value:    [abc]
Database:           default
Table:              part_table
CreateTime:         Wed Mar 30 16:57:14 PDT 2016
LastAccessTime:     UNKNOWN
Protect Mode:       None
Location:           file:/tmp/warehouse/part_table/d=abc
Partition Parameters:
    COLUMN_STATS_ACCURATE  true
    numFiles               1
    numRows                1
    rawDataSize            1
    totalSize              2
    transient_lastDdlTime  1459382234

# Storage Information
SerDe Library:      org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:        org.apache.hadoop.mapred.TextInputFormat
OutputFormat:       org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:         No
Num Buckets:        -1
Bucket Columns:     []
Sort Columns:       []
Storage Desc Params:
    serialization.format    1
Time taken: 0.334 seconds, Fetched: 35 row(s)
```

Hive 2.0+: Syntax Change

 **Hive 2.0+: New syntax**

In Hive 2.0 release onward, the describe table command has a syntax change which is backward incompatible. See [HIVE-12184](#) for details.

```
DESCRIBE [EXTENDED | FORMATTED]
        [db_name.]table_name [PARTITION partition_spec] [col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$
```

- Warning: The new syntax could break current scripts.
- It no longer accepts DOT separated table_name and column_name. They would have to be SPACE-separated. DB and TABLENAME are DOT-separated. column_name can still contain DOTs for complex datatypes.
 - Optional partition_spec has to appear after the table_name but prior to the optional column_name. In the previous syntax, column_name appears in between table_name and partition_spec.

Examples:

DESCRIBE FORMATTED default.src_table PARTITION (part_col = 100) columnA;
DESCRIBE default.src_thrift lintString.\$elem\$.myint;

HCatalog and WebHCat DDL

For information about DDL in HCatalog and WebHCat, see:

- [HCatalog DDL](#) in the [HCatalog manual](#)
- [WebHCat DDL Resources](#) in the [WebHCat manual](#)

2 people like this

No labels

Powered by a free **Atlassian Confluence Open Source Project License** granted to Apache Software Foundation. Evaluate Confluence today.
This Confluence installation runs a Free Gliffy License - Evaluate the Gliffy Confluence Plugin for your Wiki!