LanguageManual DDL

Hive Data Definition Language

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, VIEWS, PARTITIONS, FUNCTIONS, INDEX[ES], COLUMNS, CREATE TABLE
- DESCRIBE DATABASE/SCHEMA, table_name, view_name, materialized_view_name

PARTITION statements are usually options of TABLE statements, except for SHOW PARTITIONS.

Keywords, Non-reserved Keywords and Reserved Keywords

	All Keywords	
Version	**Non-reserved Keywords**	**Reserved Keywords**
Hive 1.2.0	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, METADATA, 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, RELOAD, RENAME, REPAIR, REPLACE, REPLICATION, RESTRICT, REWRITE, RLIKE, 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	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, REVOKE, RIGHT, ROLLUP, ROW, ROWS, SELECT, SET, SMALLINT, TABLE, TABLESAMPLE, THEN, TIMESTAMP, TO, TRANSFORM, TRIGGER, TRUE, TRUNCATE, UNBOUNDED, UNION, UNIQUEJOIN, UPDATE, USER, USING, UTC_TMESTAMP, VALUES, VARCHAR, WHEN, WHERE, WINDOW, WITH

Hive 2.0.0	*removed:* `REGEXP, RLIKE` *added:* `AUTOCOMMIT, ISOLATION, LEVEL, OFFSET, SNAPSHOT, TRANSACTION, WORK, WRITE`	*added:* `COMMIT, ONLY, REGEXP, RLIKE, ROLLBACK, START`
Hive 2.1.0	*added:* `ABORT, KEY, LAST, NORELY, NOVALIDATE, NULLS, RELY, VALIDATE`	*added:* `CACHE, CONSTRAINT, FOREIGN, PRIMARY, REFERENCES`
Hive 2.2.0	*added:* `DETAIL, DOW, EXPRESSION, OPERATOR, QUARTER, SUMMARY, VECTORIZATION, WEEK, YEARS, MONTHS, WEEKS, DAYS, HOURS, MINUTES, SECONDS`	*added:* `DAYOFWEEK, EXTRACT, FLOOR, INTEGER, PRECISION, VIEWS`
Hive 3.0.0	*added:* `TIMESTAMPTZ, ZONE`	*added:* `TIME, NUMERIC, SYNC`

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 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. (version 2.1.0 and earlier)

Create/Drop/Alter/Use Database

Create Database

```
CREATE [REMOTE] (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
  [COMMENT database_comment]
  [LOCATION hdfs_path]
  [MANAGEDLOCATION 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).

MANAGEDLOCATION was added to database in Hive 4.0.0 (HIVE-22995). LOCATION now refers to the default directory for external tables and MANAGEDLOCATION refers to the default directory for managed tables. Its recommended that MANAGEDLOCATION be within metastore.warehouse.dir so all managed tables have a common root where common governance policies. It can be used with metastore.warehouse.tenant.colocation to have it point to a directory outside the warehouse root directory to have a tenant based common root where quotas and other policies can be set.

REMOTE databases were added in Hive 4.0.0 (HIVE-24396) for support for Data connectors. See documentation for Data connectors.

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). The default behavior is RESTRICT, where DROP DATABASE will fail if the database is not empty. To drop the tables in the database as well, use DROP DATABASE ... CASCADE. Support for RESTRICT and CASCADE was added in Hive 0.8 (HIVE-2090).

Alter Database

```
ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES (property_name=property_value, ...);   -- (Note: SCHEMA
added in Hive 0.14.0)

ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;   -- (Note: Hive 0.13.0 and later;
SCHEMA added in Hive 0.14.0)

ALTER (DATABASE|SCHEMA) database_name SET LOCATION hdfs_path; -- (Note: Hive 2.2.1, 2.4.0 and later)

ALTER (DATABASE|SCHEMA) database_name SET MANAGEDLOCATION hdfs_path; -- (Note: Hive 4.0.0 and later)
```

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

The ALTER DATABASE ... SET LOCATION statement does not move the contents of the database's current directory to the newly specified location. It does not change the locations associated with any tables/partitions under the specified database. It only changes the default parent-directory where new tables will be added for this database. This behaviour is analogous to how changing a table-directory does not move existing partitions to a different location.

The ALTER DATABASE ... SET MANAGEDLOCATION statement does not move the contents of the database's managed tables directories to the newly specified location. It does not change the locations associated with any tables/partitions under the specified database. It only changes the default parent-directory where new tables will be added for this database. This behaviour is analogous to how changing a table-directory does not move existing partitions to a different location.

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/Alter Connector

Create Connector

```
CREATE CONNECTOR [IF NOT EXISTS] connector_name
  [TYPE datasource_type]
  [URL datasource_url]
  [COMMENT connector_comment]
  [WITH DCPROPERTIES (property_name=property_value, ...)];
```

Since Hive 4.0.0 via HIVE-24396 Support for Data connectors was added in hive 4.0.0. Initial commit includes connector implementations for JDBC based datasource like MYSQL, POSTGRES, DERBY. Additional connector implementations will be added via followup commits.

TYPE - Type of the remote datasource this connector connects to. for example MYSQL. The type determines the Driver class and any other params specific to this datasource.

URL - URL of the remote datasource. In case of JDBC datasource, it would be the JDBC connection URL. For hive types, it would be the thrift URL.

COMMENT - A short description for this connector.

DCPROPERTIES: Contains a set of name/value pairs that are set for the connector. The credentials for the remote datasource are specified as part of the DCPROPERTIES as documented in the JDBC Storage Handler docs. All properties that start with a prefix of "hive.sql" are added to the tables mapped by this connector.

Drop Connector

```
DROP CONNECTOR [IF EXISTS] connector_name;
```

Since Hive 4.0.0 via HIVE-24396. If there are databases that are mapped by this connector, drop still succeeds. Users will see errors when running DDLs like "show tables" in the mapped databases.

Alter Connector

```
ALTER CONNECTOR connector_name SET DCPROPERTIES (property_name=property_value, ...);

ALTER CONNECTOR connector_name SET URL new_url;

ALTER CONNECTOR connector_name SET OWNER [USER|ROLE] user_or_role;
```

Since Hive 4.0.0 via HIVE-24396

The ALTER CONNECTOR ... SET DCPROPERTIES replaces the existing properties with the new set of properties specified in the ALTER DDL.

The ALTER CONNECTOR ... SET URL replaces the existing URL with a new URL for the remote datasource. Any REMOTE databases that were created using the connector will continue to work as they are associated by name.

The ALTER CONNECTOR ... SET OWNER changes the ownership of the connector object in hive.

Create/Drop/Truncate Table

Create Table

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name    -- (Note: TEMPORARY available in
Hive 0.14.0 and later)
  [(col_name data_type [column_constraint_specification] [COMMENT col_comment], ... [constraint_specification])]
  [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 later)
  ]
  [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
  | DOUBLE PRECISION -- (Note: Available in Hive 2.2.0 and later)
  | 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)
  | JSONFILE    -- (Note: Available in Hive 4.0.0 and later)
  | INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

column_constraint_specification:
  : [ PRIMARY KEY|UNIQUE|NOT NULL|DEFAULT [default_value]|CHECK  [check_expression] ENABLE|DISABLE NOVALIDATE
RELY/NORELY ]

default_value:
  : [ LITERAL|CURRENT_USER()|CURRENT_DATE()|CURRENT_TIMESTAMP()|NULL ]

constraint_specification:
  : [, PRIMARY KEY (col_name, ...) DISABLE NOVALIDATE RELY/NORELY ]
    [, PRIMARY KEY (col_name, ...) DISABLE NOVALIDATE RELY/NORELY ]
    [, CONSTRAINT constraint_name FOREIGN KEY (col_name, ...) REFERENCES table_name(col_name, ...) DISABLE
NOVALIDATE
    [, CONSTRAINT constraint_name UNIQUE (col_name, ...) DISABLE NOVALIDATE RELY/NORELY ]
    [, CONSTRAINT constraint_name CHECK [check_expression] ENABLE|DISABLE NOVALIDATE RELY/NORELY ]
```

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), however, dot (**.**) and colon (**:**) yield errors on querying, so they are disallowed in Hive 1.2.0 (see HIVE-10120). 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).
- A table created without the EXTERNAL clause is called a *managed table* because Hive manages its data. To find out if a table is managed or external, look for tableType in the output of DESCRIBE EXTENDED table_name.
- 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 ("compactor.mapreduce.map.memory.mb"="*mapper_memory*") – see Hive Transactions.
    - TBLPROPERTIES ("compactorthreshold.hive.compactor.delta.num.threshold"="*threshold_num*") – see Hive Transactions.
    - TBLPROPERTIES ("compactorthreshold.hive.compactor.delta.pct.threshold"="*threshold_pct*") – see Hive Transactions.
    - TBLPROPERTIES ("auto.purge"="true") or ("auto.purge"="false") in release 1.2.0+ (HIVE-9118) – see Drop Table, Drop Partitions, Truncate Table, and Insert Overwrite.
    - TBLPROPERTIES ("EXTERNAL"="TRUE") in release 0.6.0+ (HIVE-1329) – Change a managed table to an external table and vice versa for "FALSE".
        - As of Hive 2.4.0 (HIVE-16324) the value of the property 'EXTERNAL' is parsed as a boolean (case insensitive true or false) instead of a case sensitive string comparison.
    - TBLPROPERTIES ("external.table.purge"="true") in release 4.0.0+ (HIVE-19981) when set on external table would delete the data as well.
- 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.

Managed and External Tables

By default Hive creates managed tables, where files, metadata and statistics are managed by internal Hive processes. For details on the differences between managed and external table see Managed vs. External Tables.

Storage Formats

Hive supports built-in and custom-developed file formats. See CompressedStorage for details on compressed table storage.
The following are some of the formats built-in to Hive:

Storage Format	Description
STORED AS TEXTFILE	Stored as plain text files. TEXTFILE is the default file format, unless the configuration parameter hive.default.fileformat has a different setting. Use the DELIMITED clause to read delimited files. 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'). (Hive 4.0) All BINARY columns in the table are assumed to be base64 encoded. To read the data as raw bytes: TBLPROPERTIES ("hive.serialization.decode.binary.as.base64"="false")
STORED AS SEQUENCEFILE	Stored as compressed Sequence File.
STORED AS ORC	Stored as ORC file format. Supports ACID Transactions & Cost-based Optimizer (CBO). Stores column-level metadata.
STORED AS PARQUET	Stored as Parquet format for the Parquet columnar storage format in Hive 0.13.0 and later; Use ROW FORMAT SERDE ... STORED AS INPUTFORMAT ... OUTPUTFORMAT syntax ... in Hive 0.10, 0.11, or 0.12.
STORED AS AVRO	Stored as Avro format in Hive 0.14.0 and later (see Avro SerDe).
STORED AS RCFILE	Stored as Record Columnar File format.
STORED AS JSONFILE	Stored as Json file format in Hive 4.0.0 and later.
STORED BY	Stored by a non-native table format. To create or link to a non-native table, for example a table backed by HBase or Druid or Accumulo. See StorageHandlers for more information on this option.
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).

Row Formats & 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.
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.

For general information about SerDes, see Hive SerDe in the Developer Guide. Also see SerDe for details about input and output processing.

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

Row Format	Description

RegEx
 	
ROW FORMAT SERDE
'org.apache.hadoop.hive.
serde2.RegexSerDe'
WITH SERDEPROPERTIES
(
"input.regex" = "<regex>"
)
STORED AS TEXTFILE;
	Stored as plain text file, translated by Regular Expression.
	
	The following example defines a table in the default Apache Weblog format.
	
	`CREATE TABLE apachelog (`
	`  host STRING,`
	`  identity STRING,`
	`  user STRING,`
	`  time STRING,`
	`  request STRING,`
	`  status STRING,`
	`  size STRING,`
	`  referer STRING,`
	`  agent STRING)`
	`ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'`
	`WITH SERDEPROPERTIES (`
	`  "input.regex" = "([^]*) ([^]*) ([^]*) (-|\\[^\\]*\\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^ \"]*|\".*\") ([^ \"]*|\".*\"))?"`
	`)`
	`STORED AS TEXTFILE;`
	More about RegexSerDe can be found here in HIVE-662 and HIVE-1719.

JSON
 	
ROW FORMAT SERDE
'org.apache.hive.hcatalog.
data.JsonSerDe'
STORED AS TEXTFILE
	Stored as plain text file in JSON format.
	
	The JsonSerDe for JSON files is available in Hive 0.12 and later.
	
	In some distributions, a reference to hive-hcatalog-core.jar is required.
	`ADD JAR /usr/lib/hive-hcatalog/lib/hive-hcatalog-core.jar;`
	
	`CREATE TABLE my_table(a string, b bigint, ...)`
	`ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'`
	`STORED AS TEXTFILE;`
	
	The JsonSerDe was moved to Hive from HCatalog and before it was in hive-contrib project. It was added to the Hive distribution by HIVE-4895.
	An Amazon SerDe is available at `s3://elasticmapreduce/samples/hive-ads/libs/jsonserde.jar` for releases prior to 0.12.0.
	
	The JsonSerDe for JSON files is available in Hive 0.12 and later.
	Starting in Hive 3.0.0, JsonSerDe is added to Hive Serde as "org.apache.hadoop.hive.serde2.JsonSerDe" (HIVE-19211).
	`CREATE TABLE my_table(a string, b bigint, ...)`
	`ROW FORMAT SERDE 'org.apache.hive.serde2.JsonSerDe'`
	`STORED AS TEXTFILE;`
	
	Or `STORED AS JSONFILE` is supported starting in Hive 4.0.0 (HIVE-19899), so you can create table as follows:
	
	`CREATE TABLE my_table(a string, b bigint, ...) STORED AS JSONFILE;`

CSV/TSV ROW FORMAT SERDE 'org.apache.hadoop.hive. serde2.OpenCSVSerde' STORED AS TEXTFILE	Stored as plain text file in CSV / TSV format. The CSVSerde is available in Hive 0.14 and greater. The following example creates a TSV (Tab-separated) file.

```
CREATE TABLE my_table(a string, b string, ...)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   "separatorChar" = "\t",
   "quoteChar"  = "'",
   "escapeChar"  = "\\"
)
STORED AS TEXTFILE;
```

Default properties for SerDe is Comma-Separated (CSV) file

```
DEFAULT_ESCAPE_CHARACTER \
DEFAULT_QUOTE_CHARACTER  "
DEFAULT_SEPARATOR        ,
```

This SerDe works for most CSV data, but does not handle embedded newlines. To use the SerDe, specify the fully qualified class name org.apache.hadoop.hive.serde2.OpenCSVSerde.

Documentation is based on original documentation at https://github.com/ogrodnek/csv-serde.

Limitations

This SerDe treats all columns to be of type String. Even if you create a table with non-string column types using this SerDe, the DESCRIBE TABLE output would show string column type.
The type information is retrieved from the SerDe.

To convert columns to the desired type in a table, you can create a view over the table that does the CAST to the desired type.

The CSV SerDe is based on https://github.com/ogrodnek/csv-serde, and was added to the Hive distribution in HIVE-7777.
The CSVSerde has been built and tested against Hive 0.14 and later, and uses Open-CSV 2.3 which is bundled with the Hive distribution.

For general information about SerDes, see Hive SerDe in the Developer Guide. Also see SerDe for details about input and output processing.

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. Starting Hive 4.0.0 (

> ⚠ Unable to render Jira issues macro, execution error.

) setting table property external.table.purge=true, will also delete the data.

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.

For another example of creating an external table, see Loading Data in the Tutorial.

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.

Starting with Hive 3.2.0, CTAS statements can define a partitioning specification for the target table (HIVE-20241).

CTAS has these restrictions:

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

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).

Example:

```
CREATE TEMPORARY TABLE list_bucket_multiple (col1 STRING, col2 int, col3 STRING);
```

Transactional Tables

Version information

As of Hive 4.0 (HIVE-18453).

A table that supports operations with ACID semantics. See this for more details about transactional tables.

Example:

```
CREATE TRANSACTIONAL TABLE transactional_table_test(key string, value string) PARTITIONED BY(ds string) STORED
AS ORC;
```

Constraints

> **Version information**
>
> As of Hive 2.1.0 (HIVE-13290).

Hive includes support for non-validated primary and foreign key constraints. Some SQL tools generate more efficient queries when constraints are present. Since these constraints are not validated, an upstream system needs to ensure data integrity before it is loaded into Hive.

Example:

```
create table pk(id1 integer, id2 integer,
  primary key(id1, id2) disable novalidate);

create table fk(id1 integer, id2 integer,
  constraint c1 foreign key(id1, id2) references pk(id2, id1) disable novalidate);
```

> **Version information**
>
> As of Hive 3.0.0 (HIVE-16575, HIVE-18726, HIVE-18953).

Hive includes support for UNIQUE, NOT NULL, DEFAULT and CHECK constraints. Beside UNIQUE all three type of constraints are enforced.

Example:

```
create table constraints1(id1 integer UNIQUE disable novalidate, id2 integer NOT NULL,
  usr string DEFAULT current_user(), price double CHECK (price > 0 AND price <= 1000));

create table constraints2(id1 integer, id2 integer,
  constraint c1_unique UNIQUE(id1) disable novalidate);

create table constraints3(id1 integer, id2 integer,
  constraint c1_check CHECK(id1 + id2 > 0));
```

DEFAULT on complex data types such as map, struct, array is not supported.

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. Starting Hive 4.0.0 (

> Unable to render Jira issues macro, execution error.

) setting table property external.table.purge=true, will also delete the data.

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 TBLPROPERTIES 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). The rows will be trashed if the filesystem Trash is enabled, otherwise they are deleted (as of Hive 2.2.0 with HIVE-14626). 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) if the table property "auto.purge" (see TBLPROPERTIES 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. This is applicable only for managed tables (see managed tables). This behavior can be turned off if the "auto.purge" property is unset or set to false for a managed table.

Starting with Hive 4.0 (HIVE-23183) the TABLE token is optional, previous versions required it.

Alter Table/Partition/Column

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. Rename has been changed as of version 2.2.0 (HIVE-14909) so that a managed table's HDFS location is moved only if the table is created without a LOCATION clause 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.

For more information, see the TBLPROPERTIES clause in Create Table above.

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' = ',');
```

Remove SerDe Properties

Version information

Remove SerDe Properties is supported as of Hive 4.0.0 (HIVE-21952).

```
ALTER TABLE table_name [PARTITION partition_spec] UNSET SERDEPROPERTIES (property_name, ... );
```

These statements enable you to remove user-defined metadata to the table's SerDe object.

Note that `property_name` must be quoted.

Example:

```
ALTER TABLE table_name UNSET 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.

Alter Table Constraints

> **Version information**
>
> As of Hive release 2.1.0.

Table constraints can be added or removed via ALTER TABLE statements.

```
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 ADD CONSTRAINT constraint_name UNIQUE (column, ...) DISABLE NOVALIDATE;
ALTER TABLE table_name CHANGE COLUMN column_name column_name data_type CONSTRAINT constraint_name NOT NULL
ENABLE;
ALTER TABLE table_name CHANGE COLUMN column_name column_name data_type CONSTRAINT constraint_name DEFAULT
default_value ENABLE;
ALTER TABLE table_name CHANGE COLUMN column_name column_name data_type CONSTRAINT constraint_name CHECK
check_expression ENABLE;

ALTER TABLE table_name DROP CONSTRAINT constraint_name;
```

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 'location'][, PARTITION
partition_spec [LOCATION 'location'], ...];

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+.

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

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.

Discover Partitions

Table property "discover.partitions" can now be specified to control automatic discovery and synchronization of partition metadata in Hive Metastore.

When Hive Metastore Service (HMS) is started in remote service mode, a background thread (PartitionManagementTask) gets scheduled periodically every 300s (configurable via metastore.partition.management.task.frequency config) that looks for tables with "discover.partitions" table property set to true and performs MSCK REPAIR in sync mode. If the table is a transactional table, then Exclusive Lock is obtained for that table before performing MSCK REPAIR. With this table property, "MSCK REPAIR TABLE table_name SYNC PARTITIONS" is no longer required to be run manually.

> **Version information**
>
> As of Hive 4.0.0 (HIVE-20707).

Partition Retention

Table property "partition.retention.period" can now be specified for partitioned tables with a retention interval. When a retention interval is specified, the background thread running in HMS (refer Discover Partitions section), will check the age (creation time) of the partition and if the partition's age is older than the retention period, it will be dropped. Dropping partitions after retention period will also delete the data in that partition. For example, if an external partitioned table with 'date' partition is created with table properties "discover.partitions"="true" and "partition.retention.period"="7d" then only the partitions created in last 7 days are retained.

> **Version information**
>
> As of Hive 4.0.0 (HIVE-20707).

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) or removed from HDFS, the metastore (and hence Hive) will not be aware of these changes to partition information unless the user runs `ALTER TABLE table_name ADD/DROP PARTITION` commands on each of the newly added or removed partitions, respectively.

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

```
MSCK [REPAIR] TABLE table_name [ADD/DROP/SYNC PARTITIONS];
```

which will update metadata about partitions to the Hive metastore for partitions for which such metadata doesn't already exist. The default option for MSC command is ADD PARTITIONS. With this option, it will add any partitions that exist on HDFS but not in metastore to the metastore. The DROP PARTITIONS option will remove the partition information from metastore, that is already removed from HDFS. The SYNC PARTITIONS option is equivalent to calling both ADD and DROP PARTITIONS. See HIVE-874 and HIVE-17824 for more details. When there is a large number of untracked partitions, there is a provision to run MSCK REPAIR TABLE batch wise to avoid OOME (Out of Memory Error). By giving the configured batch size for the property **hive.msck.repair.batch.size** it can run in the batches internally. The default value of the property is zero, it means it will execute all the partitions at once. MSCK command without the REPAIR option can be used to find details about metadata mismatch metastore.

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 available 2.0.0 and later)
```

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 TBLPROPERTIES 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. The operation only changes the table metadata. Any conversion of existing data must be done outside of Hive.

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.
> As of Hive release 1.3.0 and 2.1.0 when transactions are being used, the ALTER TABLE ... COMPACT statement can include a TBLPROPERTIES clause that is either to change compaction MapReduce job properties or to overwrite any other Hive table properties. More details can be found here.
> As of Hive release 4.0.0-alpha-2 compaction pooling is available.
> As of Hive release 4.0.0 rebalance compaction is available.

```
ALTER TABLE table_name [PARTITION (partition_key = 'partition_value' [, ...])]
  COMPACT 'compaction_type'[AND WAIT]
   [CLUSTERED INTO n BUCKETS]
   [ORDER BY col_list]
   [POOL 'pool_name']
   [WITH OVERWRITE TBLPROPERTIES ("property"="value" [, ...])];
```

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. By default the statement will enqueue a request for compaction and return. To watch the progress of the compaction, use SHOW COMPACTIONS. As of Hive 2.2.0 "AND WAIT" may be specified to have the operation block until compaction completes.

The compaction_type can be MAJOR, MINOR or REBALANCE. See the Basic Design section in Hive Transactions for more information.

More in formation on compaction pooling can be found here: Compaction pooling

More in formation on rebalance compaction pooling can be found here: Rebalance Compaction

> The [CLUSTERED INTO n BUCKETS] and [ORDER BY col_list] clauses are only supported for REBALANCE compaction.

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 Table/Partition Update columns

> **Version information**
>
> In Hive release 3.0.0 this command was added to let the user sync serde stored schema information to metastore.

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

Tables that have serdes which self-describe the table schema may have different schemas in reality and the ones stored in Hive Metastore. For example when a user creates an Avro stored table using a schema url or schema literal, the schema will be inserted into HMS and then will never be changed in HMS regardless of url or literal changes within the serde. This can lead to problems especially when integrating with other Apache components.

The update columns feature provides a way for the user to let any schema changes made in the serde to be synced into HMS. It works on both the table and the partitions levels, and obviously only for tables whose schema is not tracked by HMS (see metastore.serdes.using.metastore.for.schema). Using the command on these latter serde types will result in error.

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), however, dot (.) and colon (:) yield errors on querying. 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 1.1.0. 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.

> 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.

> 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 1.1.0 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 1.1.0. 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

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. 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. As of Hive 2.2.0, SHOW VIEWS displays a list of views in a database.

Version Information

Originally, the file format for views was hard coded as SequenceFile. Hive 2.1.0 (HIVE-13736) made views follow the same defaults as tables and indexes using the hive.default.fileformat and hive.default.fileformat.managed properties.

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 Materialized View

- Create Materialized View
- Drop Materialized View
- Alter Materialized View

> **Version information**
>
> Materialized view support is only available in Hive 3.0 and later.

This section provides an introduction to Hive materialized views syntax. More information about materialized view support and usage in Hive can be found here.

Create Materialized View

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [db_name.]materialized_view_name
  [DISABLE REWRITE]
  [COMMENT materialized_view_comment]
  [PARTITIONED ON (col_name, ...)]
  [CLUSTERED ON (col_name, ...) | DISTRIBUTED ON (col_name, ...) SORTED ON (col_name, ...)]
  [
    [ROW FORMAT row_format]
    [STORED AS file_format]
      | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]
  ]
  [LOCATION hdfs_path]
  [TBLPROPERTIES (property_name=property_value, ...)]
AS SELECT ...;
```

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

The names of the materialized view's columns will be derived automatically from the defining SELECT expression.

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

By default, materialized views are enabled to be used by the query optimizer for automatic rewriting when they are created.

> **Version information**
>
> PARTITIONED ON is supported as of Hive 3.2.0 (HIVE-14493).

> **Version information**

CLUSTERED/DISTRIBUTED/SORTED ON is supported as of Hive 4.0.0 (HIVE-18842).

Drop Materialized View

```
DROP MATERIALIZED VIEW [db_name.]materialized_view_name;
```

DROP MATERIALIZED VIEW removes metadata and data for this materialized view.

Alter Materialized View

Once a materialized view has been created, the optimizer will be able to exploit its definition semantics to automatically rewrite incoming queries using materialized views, and hence, accelerate query execution.

Users can selectively enable/disable materialized views for rewriting. Recall that, by default, materialized views are enabled for rewriting at creation time. To alter that behavior, the following statement can be used:

```
ALTER MATERIALIZED VIEW [db_name.]materialized_view_name ENABLE|DISABLE REWRITE;
```

Create/Drop/Alter Index

Version information

As of Hive 0.7.

Indexing Is Removed since 3.0! See Indexes design document

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 (FUNCTIONS|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 FUNCTIONS 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. Due to backward compatibility reasons RELOAD FUNCTION; is also accepted.

Create/Drop/Grant/Revoke Roles and Privileges

Hive deprecated authorization mode / 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 Connectors
- Show Tables/Views/Materialized Views/Partitions/Indexes
  - Show Tables
  - Show Views
  - Show Materialized Views
  - 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'.

> **Version information: SHOW DATABASES**
>
> Starting from 4.0.0 we accept only SQL type like expressions containing '%' for any character(s), and '_' for a single character. Examples are 'employees', 'emp%', 'emplo_ees', all of which will match the database named 'employees'.

Show Connectors

```
SHOW CONNECTORS;
```

Since Hive 4.0.0 via HIVE-24396

SHOW CONNECTORS lists all of the connectors defined in the metastore (depending on the user's access).

Show Tables/Views/Materialized Views/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 Views

> **Version information**
>
> Introduced in Hive 2.2.0 via HIVE-14558.

```
SHOW VIEWS [IN/FROM database_name] [LIKE 'pattern_with_wildcards'];
```

SHOW VIEWS lists all the views in the current database (or the one explicitly named using the IN or FROM 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' view. Matching views are listed in alphabetical order. It is not an error if no matching views are found in metastore. If no regular expression is given then all views in the selected database are listed.

Examples

```
SHOW VIEWS;                              -- show all views in the current database
SHOW VIEWS 'test_*';                     -- show all views that start with "test_"
SHOW VIEWS '*view2';                     -- show all views that end in "view2"
SHOW VIEWS LIKE 'test_view1|test_view2'; -- show views named either "test_view1" or "test_view2"
SHOW VIEWS FROM test1;                   -- show views from database test1
SHOW VIEWS IN test1;                     -- show views from database test1 (FROM and IN are same)
SHOW VIEWS IN test1 "test_*";            -- show views from database test2 that start with "test_"
```

Show Materialized Views

```
SHOW MATERIALIZED VIEWS [IN/FROM database_name] [LIKE 'pattern_with_wildcards’];
```

SHOW MATERIALIZED VIEWS lists all the views in the current database (or the one explicitly named using the IN or FROM clause) with names matching the optional regular expression. It also shows additional information about the materialized view, e.g., whether rewriting is enabled, and the refresh mode for the materialized view. Wildcards in the regular expression can only be '*' for any character(s) or '|' for a choice. If no regular expression is given then all materialized views 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)
```

Version information

Starting with Hive 4.0.0, SHOW PARTITIONS can optionally use the WHERE/ORDER BY/LIMIT clause to filter/order/limit the resulting list (HIVE-22458). These clauses work in a similar way as they do in a SELECT statement.

```
SHOW PARTITIONS [db_name.]table_name [PARTITION(partition_spec)] [WHERE where_condition] [ORDER BY col_list]
[LIMIT rows];   -- (Note: Hive 4.0.0 and later)
```

Example:

```
SHOW PARTITIONS databaseFoo.tableBar LIMIT 10;                                                    --
(Note: Hive 4.0.0 and later)
SHOW PARTITIONS databaseFoo.tableBar PARTITION(ds='2010-03-03') LIMIT 10;                         --
(Note: Hive 4.0.0 and later)
SHOW PARTITIONS databaseFoo.tableBar PARTITION(ds='2010-03-03') ORDER BY hr DESC LIMIT 10;        --
(Note: Hive 4.0.0 and later)
SHOW PARTITIONS databaseFoo.tableBar PARTITION(ds='2010-03-03') WHERE hr >= 10 ORDER BY hr DESC LIMIT 10;    --
(Note: Hive 4.0.0 and later)
SHOW PARTITIONS databaseFoo.tableBar WHERE hr >= 10 AND ds='2010-03-03' ORDER BY hr DESC LIMIT 10;  --
(Note: Hive 4.0.0 and later)
```

Note: Please use *hr >= 10* instead of *hr - 10 >= 0* to filter the results, as Metastore would not push the latter predicate down into the underlying storage.

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.

For more information, see the TBLPROPERTIES clause in Create Table above.

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.
>
> Indexing Is Removed since 3.0! See Indexes design document

```
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.

> **Version information**
>
> `SHOW COLUMNS (FROM|IN) table_name [(FROM|IN) db_name]  [ LIKE 'pattern_with_wildcards'];`
>
> Added in Hive 3.0 by HIVE-18373.

`SHOW COLUMNS` lists all the columns in the table 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 'cola', 'col*', '*a|col*', all which will match the 'cola' column. Matching columns are listed in alphabetical order. It is not an error if no matching columns are found in table. If no regular expression is given then all columns in the selected table are listed.

Examples

```
-- SHOW COLUMNS
CREATE DATABASE test_db;
USE test_db;
CREATE TABLE foo(col1 INT, col2 INT, col3 INT, cola INT, colb INT, colc INT, a INT, b INT, c INT);

-- SHOW COLUMNS basic syntax
SHOW COLUMNS FROM foo;                          -- show all column in foo
SHOW COLUMNS FROM foo "*";                      -- show all column in foo
SHOW COLUMNS IN foo "col*";                     -- show columns in foo starting with "col"
OUTPUT col1,col2,col3,cola,colb,colc
SHOW COLUMNS FROM foo '*c';                     -- show columns in foo ending with "c"
OUTPUT c,colc
SHOW COLUMNS FROM foo LIKE "col1|cola";         -- show columns in foo either col1 or cola
OUTPUT col1,cola
SHOW COLUMNS FROM foo FROM test_db LIKE 'col*';   -- show columns in foo starting with "col"
OUTPUT col1,col2,col3,cola,colb,colc
SHOW COLUMNS IN foo IN test_db LIKE 'col*';       -- show columns in foo starting with "col" (FROM/IN same)
OUTPUT col1,col2,col3,cola,colb,colc

-- Non existing column pattern resulting in no match
SHOW COLUMNS IN foo "nomatch*";
SHOW COLUMNS IN foo "col+";                     -- + wildcard not supported
SHOW COLUMNS IN foo "nomatch";
```

Show Functions

```
SHOW FUNCTIONS [LIKE "<pattern>"];
```

SHOW FUNCTIONS lists all the user defined and builtin functions, filtered by the the regular expression if specified with LIKE.

Show Granted Roles and Privileges

Hive deprecated authorization mode / 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
- agent info – a string that helps identify the entity that issued the lock request. For a SQL client this is the query ID, for streaming client it may be Storm bolt ID for example.

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
- timestamp when the transaction was started (as of Hive 2.2.0)
- timestamp for last heartbeat (as of Hive 2.2.0)

Show Compactions

> **Version information**
>
> As of Hive 0.13.0 (see Hive Transactions).

```
SHOW COMPACTIONS [DATABASE.][TABLE] [PARTITION (<partition_spec>)] [POOL_NAME] [TYPE] [STATE] [ORDER BY `start` DESC] [LIMIT 10];
```

SHOW COMPACTIONS returns a list of all compaction requests currently being processed or scheduled, including this information:

- "CompactionId" - unique internal id (As of Hive 3.0)

- "Database" - Hive database name
- "Table" - table name
- "Partition" - partition name (if the table is partitioned)
- "Type" - whether it is a major or minor compaction
- "State" - 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
    - "failed" – the job failed. The metastore log will have more detail.
    - "succeeded" – A-ok
    - "attempted" – initiator attempted to schedule a compaction but failed. The metastore log will have more information.
- "Worker" - thread ID of the worker thread doing the compaction (only if in working state)
- "Start Time" - the time at which the compaction started (only if in working or ready for cleaning state)
- "Duration(ms)" - time this compaction took (As of Hive 2.2)
- "HadoopJobId" - Id of the submitted Hadoop job (As of Hive 2.2)
- “Enqueue Time” - Time spent by compaction before start
- “Initiator host”- Host ID which started compaction
- “TxnId” - A transaction Id associated with this compaction
- “Commit Time” - Total time taken by compaction
- “Highest WriteId” Highest writeId that compactor includes
- “Pool name”- A pool associated with given compaction or default if not associated
- “Error message”- error message if any

Examples:

```
Examples
SHOW COMPACTIONS.
 – show all compactions of all tables and partitions currently being compacted or scheduled  for compaction

SHOW COMPACTIONS DATABASE db1
 – show all compactions of all tables from given database which are currently being compacted or scheduled for
compaction

SHOW COMPACTIONS SCHEMA db1
 – show all compactions of all tables from given database which are currently being compacted or scheduled for
compaction

SHOW COMPACTIONS tbl0
 – show all compactions from given table which are currently being compacted or scheduled for compaction

SHOW COMPACTIONS compactionid =1
 – show all compactions with given compaction ID

SHOW COMPACTIONS db1.tbl0 PARTITION (p=101,day='Monday') POOL 'pool0' TYPE 'minor' STATUS 'ready for clean'
ORDER BY cq_table DESC, cq_state LIMIT 42
 – show all compactions from specific database/table filtered based on pool name/type.state/status and ordered
with given clause
```

Compactions are initiated automatically, but can also be initiated manually with an ALTER TABLE COMPACT statement.

Describe

Describe Database

> **Version information**
>
> As of Hive 0.7.

```
DESCRIBE DATABASE [EXTENDED] db_name;
DESCRIBE SCHEMA [EXTENDED] db_name;     -- (Note: Hive 1.1.0 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 1.1.0 (HIVE-8803).

EXTENDED also shows the database properties.

Describe Dataconnector

```
DESCRIBE CONNECTOR [EXTENDED] connector_name;
```

Since Hive 4.0.0 via HIVE-24396

DESCRIBE CONNECTOR shows the name of the connector, its comment (if one has been set), and its datasource URL and datasource type.

EXTENDED also shows the dataconnector's properties. Any clear-text passwords set will be shown in clear text as well.

Describe Table/View/Materialized View/Column

There are two formats for the describe table/view/materialized 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, then it will show the metadata in a tabular format.

Note: DESCRIBE EXTENDED shows the number of rows only if statistics were gathered when the data was loaded (see Newly Created Tables), and if the Hive CLI is used instead of a Thrift client or Beeline. HIVE-6285 will address this issue. Although ANALYZE TABLE gathers statistics after the data has been loaded (see Existing Tables), it does not currently provide information about the number of rows.

If a table has a complex column then you can examine the attributes of this column by specifying table_name.complex_col_name (and field_name for an element of a struct, '$elem$' for array element, 'key' for map key, and '$value$' for map value). You can specify this recursively to explore the complex column type.

For a view, DESCRIBE EXTENDED or FORMATTED can be used to retrieve the view's definition. Two relevant attributes are provided: both the original view definition as specified by the user, and an expanded definition used internally by Hive.

For materialized views, DESCRIBE EXTENDED or FORMATTED provides additional information on whether rewriting is enabled and whether the given materialized view is considered to be up-to-date for automatic rewriting with respect to the data in the source tables that it uses.

Version information — partition & non-partition columns

In Hive 0.10.0 and earlier, no distinction is made between partition columns and non-partition columns while displaying columns for DESCRIBE TABLE. From Hive 0.12.0 onwards, they are displayed separately.

In Hive 0.13.0 and later, the configuration parameter hive.display.partition.cols.separately lets you use the old behavior, if desired (HIVE-6689). For an example, see the test case in the patch for HIVE-6689.

Bug fixed in Hive 0.10.0 — database qualifiers

Database qualifiers for table names were introduced in Hive 0.7.0, but they were broken for DESCRIBE until a bug fix in Hive 0.10.0 (HIVE-1977).

Bug fixed in Hive 0.13.0 — quoted identifiers

Prior to Hive 0.13.0 DESCRIBE did not accept backticks (`) surrounding table identifiers, so DESCRIBE could not be used for tables with names that matched reserved keywords (HIVE-2949 and HIVE-6187). As of 0.13.0, all identifiers specified within backticks are treated literally when the configuration parameter hive.support.quoted.identifiers has its default value of "column" (HIVE-6013). The only exception is that double backticks (``) represent a single backtick character.

Display Column Statistics

Version information

As of Hive 0.14.0; see HIVE-7050 and HIVE-7051. (The FOR COLUMNS option of ANALYZE TABLE is available as of Hive 0.10.0.)

ANALYZE TABLE *table_name* COMPUTE STATISTICS FOR COLUMNS will compute column statistics for all columns in the specified table (and for all partitions if the table is partitioned). To view the gathered column statistics, the following statements can be used:

```
DESCRIBE FORMATTED [db_name.]table_name column_name;                                  -- (Note: Hive 0.14.0 and
later)
DESCRIBE FORMATTED [db_name.]table_name column_name PARTITION (partition_spec);   -- (Note: Hive 0.14.0 to 1.x.
x)
                                                                                      -- (see "Hive 2.0+: New
Syntax" below)
```

See Statistics in Hive: Existing Tables for more information about the ANALYZE TABLE command.

Describe Partition

There are two formats for the describe partition 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[.column_name] PARTITION partition_spec;
                                      -- (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 [column_name] PARTITION partition_spec;
                                      -- (Note: Hive 1.x.x and 0.x.x only. See "Hive 2.0+: New Syntax" below)
```

This statement lists metadata for a given partition. The output is similar to that of DESCRIBE table_name. Presently, the column information associated with a particular partition is not used while preparing plans. As of Hive 1.2 (HIVE-10307), the partition column values specified in *partition_spec* are type validated, converted and normalized to their column types when hive.typecheck.on.insert is set to true (default). These values can be number literals.

Example:

```
hive> show partitions part_table;
OK
d=abc


hive> DESCRIBE extended part_table partition (d='abc');
OK
i                       int
d                       string

# Partition Information
# col_name              data_type               comment

d                       string

Detailed Partition Information  Partition(values:[abc], dbName:default, tableName:part_table, createTime:
1459382234, lastAccessTime:0, sd:StorageDescriptor(cols:[FieldSchema(name:i, type:int, comment:null),
FieldSchema(name:d, type:string, comment:null)], location:file:/tmp/warehouse/part_table/d=abc, inputFormat:org.
apache.hadoop.mapred.TextInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat,
compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.
lazy.LazySimpleSerDe, parameters:{serialization.format=1}), bucketCols:[], sortCols:[], parameters:{},
skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}),
```

```
storedAsSubDirectories:false), parameters:{numFiles=1, COLUMN_STATS_ACCURATE=true,
transient_lastDdlTime=1459382234, numRows=1, totalSize=2, rawDataSize=1})
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;
```

Abort

- Abort Transactions

Abort Transactions

> **Version information**
>
> As of Hive 1.3.0 and 2.1.0 (see Hive Transactions).

```
ABORT TRANSACTIONS transactionID [ transactionID ...];
```

ABORT TRANSACTIONS cleans up the specified transaction IDs from the Hive metastore so that users do not need to interact with the metastore directly in order to remove dangling or failed transactions. ABORT TRANSACTIONS is added in Hive 1.3.0 and 2.1.0 (HIVE-12634).

Example:

```
ABORT TRANSACTIONS 0000007 0000008 0000010 0000015;
```

This command can be used together with SHOW TRANSACTIONS. The latter can help figure out the candidate transaction IDs to be cleaned up.

Scheduled queries

Documentation is available on the Scheduled Queries page.

Datasketches integration

Documentation is available on the Datasketches Integration page

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