Hive Data Definitions

HiveQL is the Hive query language. Like all SQL dialects in widespread use, it doesn’t fully conform to any particular revision of the ANSI SQL standard. It is perhaps closest to MySQL’s dialect, but with significant differences. Hive offers no support for row-level inserts, updates, and deletes. Hive doesn’t support transactions. Hive adds extensions to provide better performance in the context of Hadoop and to integrate with custom extensions and even external programs.

Databases in Hive

The Hive concept of a database is essentially just a *catalog* or *namespace* of tables. However, they are very useful for larger clusters with multiple teams and users, as a way of avoiding table name collisions. It’s also common to use databases to organize production tables into logical groups.

If you don’t specify a database, the default database is used.

The simplest syntax for creating a database is shown in the following example:

hive> **CREATE** **DATABASE** financials;

Hive will throw an error if financials already exists. You can suppress these warnings with this variation:

hive> **CREATE** **DATABASE** IF **NOT** **EXISTS** financials;

While normally you might like to be warned if a database of the same name already exists, the IF NOT EXISTS clause is useful for scripts that should create a database on-the-fly, if necessary, before proceeding.

You can also use the keyword SCHEMA instead of DATABASE in all the database-related commands.

At any time, you can see the databases that already exist as follows:

hive> **SHOW** DATABASES;

**default**

financials

hive> **CREATE** **DATABASE** human\_resources;

hive> **SHOW** DATABASES;

**default**

financials

human\_resources

If you have a lot of databases, you can restrict the ones listed using a *regular expression*, a concept we’ll explain in [LIKE and RLIKE](https://www.safaribooksonline.com/library/view/programming-hive/9781449326944/ch06.html#LIKE-RLIKE), if it is new to you. The following example lists only those databases that start with the letter h and end with any other characters (the .\* part):

hive> **SHOW** DATABASES **LIKE** 'h.\*';

human\_resources

hive> ...

Hive will create a directory for each database. Tables in that database will be stored in subdirectories of the database directory. The exception is tables in the default database, which doesn’t have its own directory.

The database directory is created under a top-level directory specified by the property hive.metastore.warehouse.dir, which we discussed in [Local Mode Configuration](https://www.safaribooksonline.com/library/view/programming-hive/9781449326944/ch02.html#Local-Mode-Configuration) and [Distributed and Pseudodistributed Mode Configuration](https://www.safaribooksonline.com/library/view/programming-hive/9781449326944/ch02.html#Distributed-and-Pseudo-Distributed-Mode-Configuration). Assuming you are using the default value for this property,*/user/hive/warehouse*, when the financials database is created, Hive will create the directory */user/hive/warehouse/financials.db*. Note the *.db*extension.

You can override this default location for the new directory as shown in this example:

hive> **CREATE** **DATABASE** financials

> **LOCATION** '/my/preferred/directory';

You can add a descriptive comment to the database, which will be shown by the DESCRIBE DATABASE <database> command.

hive> **CREATE** **DATABASE** financials

> **COMMENT** 'Holds all financial tables';

hive> **DESCRIBE** **DATABASE** financials;

financials Holds **all** financial tables

hdfs://master-server/**user**/hive/warehouse/financials.db

The USE command sets a database as your working database, analogous to changing working directories in a filesystem:

hive> USE financials;

Now, commands such as SHOW TABLES; will list the tables in this database.

Unfortunately, there is no command to show you which database is your current working database! Fortunately, it’s always safe to repeat the USE …command; there is no concept in Hive of nesting of databases.

Recall that we pointed out a useful trick in [Variables and Properties](https://www.safaribooksonline.com/library/view/programming-hive/9781449326944/ch02.html#VariablesAndProperties) for setting a property to print the current database as part of the prompt (Hive v0.8.0 and later):

hive> **set** hive.cli.print.**current**.db=**true**;

hive (financials)> USE **default**;

hive (**default**)> **set** hive.cli.print.**current**.db=**false**;

hive> ...

Finally, you can drop a database:

hive> **DROP** **DATABASE** IF **EXISTS** financials;

The IF EXISTS is optional and suppresses warnings if financials doesn’t exist.

By default, Hive won’t permit you to drop a database if it contains tables. You can either drop the tables first or append the CASCADE keyword to the command, which will cause the Hive to drop the tables in the database first:

hive> **DROP** **DATABASE** IF **EXISTS** financials **CASCADE**;

Using the RESTRICT keyword instead of CASCADE is equivalent to the default behavior, where existing tables must be dropped before dropping the database.

When a database is dropped, its directory is also deleted.

# Alter Database

You can set key-value pairs in the DBPROPERTIES associated with a database using the ALTER DATABASE command. No other metadata about the database can be changed, including its name and directory location:

hive> **ALTER** **DATABASE** financials **SET** DBPROPERTIES ('edited-by' = 'Joe Dba');

There is no way to delete or “unset” a DBPROPERTY.

# Creating Tables

The CREATE TABLE statement follows SQL conventions, but Hive’s version offers significant extensions to support a wide range of flexibility where the data files for tables are stored, the formats used, etc. We discussed many of these options in [Text File Encoding of Data Values](https://www.safaribooksonline.com/library/view/programming-hive/9781449326944/ch03.html#TextFileEncodingOfDataValues) and we’ll return to more advanced options later in [Chapter 15](https://www.safaribooksonline.com/library/view/programming-hive/9781449326944/ch15.html). In this section, we describe the other options available for the CREATE TABLE statement, adapting the employeestable declaration we used previously in [Collection Data Types](https://www.safaribooksonline.com/library/view/programming-hive/9781449326944/ch03.html#Collection-Data-Types):

**CREATE** **TABLE** IF **NOT** **EXISTS** mydb.employees (

name STRING **COMMENT** 'Employee name',

salary FLOAT **COMMENT** 'Employee salary',

subordinates ARRAY<STRING> **COMMENT** 'Names of subordinates',

deductions **MAP**<STRING, FLOAT>

**COMMENT** 'Keys are deductions names, values are percentages',

address STRUCT<street:STRING, city:STRING, **state**:STRING, zip:INT>

**COMMENT** 'Home address')

**COMMENT** 'Description of the table'

TBLPROPERTIES ('creator'='me', 'created\_at'='2012-01-02 10:00:00', ...)

**LOCATION** '/user/hive/warehouse/mydb.db/employees';

First, note that you can prefix a database name, mydb in this case, if you’re not currently working in the target database.

If you add the option IF NOT EXISTS, Hive will silently ignore the statement if the table already exists. This is useful in scripts that should create a table the first time they run.

However, the clause has a gotcha you should know. If the schema specified differs from the schema in the table that already exists, Hive won’t warn you. If your intention is for this table to have the new schema, you’ll have to drop the old table, losing your data, and then re-create it. Consider if you should use one or more ALTER TABLE statements to change the existing table schema instead.

**● Hive Data Manipulations**

### Loading files into tables

Hive does not do any transformation while loading data into tables. Load operations are currently pure copy/move operations that move datafiles into locations corresponding to Hive tables.

##### Syntax

|  |
| --- |
| LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)] |

##### Synopsis

Load operations are currently pure copy/move operations that move datafiles into locations corresponding to Hive tables.

* filepath can be:
  + a relative path, such as project/data1
  + an absolute path, such as /user/hive/project/data1
  + a full URI with scheme and (optionally) an authority, such as hdfs://namenode:9000/user/hive/project/data1
* The target being loaded to can be a table or a partition. If the table is partitioned, then one must specify a specific partition of the table by specifying values for all of the partitioning columns.
* filepath can refer to a file (in which case Hive will move the file into the table) or it can be a directory (in which case Hive will move all the files within that directory into the table). In either case, filepath addresses a set of files.
* If the keyword LOCAL is specified, then:
  + the load command will look for filepath in the local file system. If a relative path is specified, it will be interpreted relative to the user's current working directory. The user can specify a full URI for local files as well - for example: [file:///user/hive/project/data1](file:///\\user\hive\project\data1)
  + the load command will try to copy all the files addressed by filepath to the target filesystem. The target file system is inferred by looking at the location attribute of the table. The copied data files will then be moved to the table.
* If the keyword LOCAL is not specified, then Hive will either use the full URI of filepath, if one is specified, or will apply the following rules:
  + If scheme or authority are not specified, Hive will use the scheme and authority from the hadoop configuration variable fs.default.namethat specifies the Namenode URI.
  + If the path is not absolute, then Hive will interpret it relative to /user/<username>
  + Hive will move the files addressed by filepath into the table (or partition)
* If the OVERWRITE keyword is used then the contents of the target table (or partition) will be deleted and replaced by the files referred to by filepath; otherwise the files referred by filepath will be added to the table.

##### Notes

* filepath cannot contain subdirectories.
* If the keyword LOCAL is not given, filepath must refer to files within the same filesystem as the table's (or partition's) location.
* Hive does some minimal checks to make sure that the files being loaded match the target table. Currently it checks that if the table is stored in sequencefile format, the files being loaded are also sequencefiles, and vice versa.
* A bug that prevented loading a file when its name includes the "+" character is fixed in release 0.13.0 ([HIVE-6048](https://issues.apache.org/jira/browse/HIVE-6048)).
* Please read [CompressedStorage](https://cwiki.apache.org/confluence/display/Hive/CompressedStorage) if your datafile is compressed.

### Inserting data into Hive Tables from queries

Query Results can be inserted into tables by using the insert clause.

##### Syntax

|  |
| --- |
| Standard syntax:  INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select\_statement1 FROM from\_statement;  INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1 FROM from\_statement;    Hive extension (multiple inserts):  FROM from\_statement  INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select\_statement1  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]] select\_statement2]  [INSERT INTO TABLE tablename2 [PARTITION ...] select\_statement2] ...;  FROM from\_statement  INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1  [INSERT INTO TABLE tablename2 [PARTITION ...] select\_statement2]  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]] select\_statement2] ...;    Hive extension (dynamic partition inserts):  INSERT OVERWRITE TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...) select\_statement FROM from\_statement;  INSERT INTO TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...) select\_statement FROM from\_statement; |

##### Synopsis

* INSERT OVERWRITE will overwrite any existing data in the table or partition
  + unless IF NOT EXISTS is provided for a partition (as of Hive [0.9.0](https://issues.apache.org/jira/browse/HIVE-2612)).
  + As of Hive 2.3.0 ([HIVE-15880](https://issues.apache.org/jira/browse/HIVE-15880)), if the table has [TBLPROPERTIES](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-listTableProperties) ("auto.purge"="true") the previous data of the table is not moved to Trash when INSERT OVERWRITE query is run against the table. This functionality is applicable only for managed tables (see [managed tables](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ManagedandExternalTables)) and is turned off when "auto.purge" property is unset or set to false.
* INSERT INTO will append to the table or partition, keeping the existing data intact. (Note: INSERT INTO syntax is only available starting in version 0.8.)
  + As of Hive [0.13.0](https://issues.apache.org/jira/browse/HIVE-6406), a table can be made **immutable** by creating it with [TBLPROPERTIES ("immutable"="true")](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateTable). The default is "immutable"="false".  
    INSERT INTO behavior into an immutable table is disallowed if any data is already present, although INSERT INTO still works if the immutable table is empty. The behavior of INSERT OVERWRITE is not affected by the "immutable" table property.  
    An immutable table is protected against accidental updates due to a script loading data into it being run multiple times by mistake. The first insert into an immutable table succeeds and successive inserts fail, resulting in only one set of data in the table, instead of silently succeeding with multiple copies of the data in the table.
* Inserts can be done to a table or a partition. If the table is partitioned, then one must specify a specific partition of the table by specifying values for all of the partitioning columns. If [hive.typecheck.on.insert](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties" \l "ConfigurationProperties-hive.typecheck.on.insert) is set to true, these values are validated, converted and normalized to conform to their column types (Hive [0.12.0](https://issues.apache.org/jira/browse/HIVE-5297) onward).
* Multiple insert clauses (also known as Multi Table Insert) can be specified in the same query.
* The output of each of the select statements is written to the chosen table (or partition). Currently the OVERWRITE keyword is mandatory and implies that the contents of the chosen table or partition are replaced with the output of corresponding select statement.
* The output format and serialization class is determined by the table's metadata (as specified via DDL commands on the table).
* As of [Hive 0.14](https://issues.apache.org/jira/browse/HIVE-5317), if a table has an OutputFormat that implements AcidOutputFormat and the system is configured to use a [transaction](https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions) manager that implements ACID, then INSERT OVERWRITE will be disabled for that table.  This is to avoid users unintentionally overwriting transaction history.  The same functionality can be achieved by using [TRUNCATE TABLE](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-TruncateTable) (for non-partitioned tables) or [DROP PARTITION](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropPartitions) followed by INSERT INTO.
* As of Hive [1.1.0](https://issues.apache.org/jira/browse/HIVE-9353) the TABLE keyword is optional.
* As of Hive [1.2.0](https://issues.apache.org/jira/browse/HIVE-9481) each INSERT INTO T can take a column list like INSERT INTO T (z, x, c1).  See Description of [HIVE-9481](https://issues.apache.org/jira/browse/HIVE-9481) for examples.

##### Notes

* Multi Table Inserts minimize the number of data scans required. Hive can insert data into multiple tables by scanning the input data just once (and applying different query operators) to the input data.
* Starting with [Hive 0.13.0](https://issues.apache.org/jira/browse/HIVE-1180), the select statement can include one or more common table expressions (CTEs) as shown in the [SELECT syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-SelectSyntax). For an example, see [Common Table Expression](https://cwiki.apache.org/confluence/display/Hive/Common+Table+Expression#CommonTableExpression-CTEinViews,CTAS,andInsertStatements).

##### Dynamic Partition Inserts

**Version information**

This information reflects the situation in Hive 0.12; dynamic partition inserts were added in Hive 0.6.

In the dynamic partition inserts, users can give partial partition specifications, which means just specifying the list of partition column names in the PARTITION clause. The column values are optional. If a partition column value is given, we call this a static partition, otherwise it is a dynamic partition. Each dynamic partition column has a corresponding input column from the select statement. This means that the dynamic partition creation is determined by the value of the input column. The dynamic partition columns must be **specified last** among the columns in the SELECT statement and **in the same order** in which they appear in the PARTITION() clause.

Dynamic partition inserts are disabled by default prior to Hive 0.9.0 and enabled by default in Hive [0.9.0](https://issues.apache.org/jira/browse/HIVE-2835) and later. These are the relevant configuration properties for dynamic partition inserts:

| **Configuration property** | **Default** | **Note** |
| --- | --- | --- |
| hive.exec.dynamic.partition | true | Needs to be set to true to enable dynamic partition inserts |
| hive.exec.dynamic.partition.mode | strict | In strict mode, the user must specify at least one static partition in case the user accidentally overwrites all partitions, in nonstrict mode all partitions are allowed to be dynamic |
| hive.exec.max.dynamic.partitions.pernode | 100 | Maximum number of dynamic partitions allowed to be created in each mapper/reducer node |
| hive.exec.max.dynamic.partitions | 1000 | Maximum number of dynamic partitions allowed to be created in total |
| hive.exec.max.created.files | 100000 | Maximum number of HDFS files created by all mappers/reducers in a MapReduce job |
| hive.error.on.empty.partition | false | Whether to throw an exception if dynamic partition insert generates empty results |

###### Example

|  |
| --- |
| FROM page\_view\_stg pvs  INSERT OVERWRITE TABLE page\_view PARTITION(dt='2008-06-08', country)         SELECT pvs.viewTime, pvs.userid, pvs.page\_url, pvs.referrer\_url, null, null, pvs.ip, pvs.cnt |

Here the country partition will be dynamically created by the last column from the SELECT clause (i.e. pvs.cnt). Note that the name is not used. In nonstrict mode the dt partition could also be dynamically created.

###### Additional Documentation

* [Design Document](https://cwiki.apache.org/confluence/display/Hive/DynamicPartitions)
  + [Original design doc](https://issues.apache.org/jira/secure/attachment/12437909/dp_design.txt)
  + [HIVE-936](https://issues.apache.org/jira/browse/HIVE-936)
* [Tutorial: Dynamic-Partition Insert](https://cwiki.apache.org/confluence/display/Hive/Tutorial#Tutorial-Dynamic-PartitionInsert)
* [HCatalog Dynamic Partitioning](https://cwiki.apache.org/confluence/display/Hive/HCatalog+DynamicPartitions)
  + [Usage with Pig](https://cwiki.apache.org/confluence/display/Hive/HCatalog+DynamicPartitions#HCatalogDynamicPartitions-UsagewithPig)
  + [Usage from MapReduce](https://cwiki.apache.org/confluence/display/Hive/HCatalog+DynamicPartitions#HCatalogDynamicPartitions-UsagefromMapReduce)

### Writing data into the filesystem from queries

Query results can be inserted into filesystem directories by using a slight variation of the syntax above:

##### Syntax

|  |
| --- |
| Standard syntax:  INSERT OVERWRITE [LOCAL] DIRECTORY directory1    [ROW FORMAT row\_format] [STORED AS file\_format] (Note: Only available starting with Hive 0.11.0)    SELECT ... FROM ...    Hive extension (multiple inserts):  FROM from\_statement  INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select\_statement1  [INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select\_statement2] ...      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: Only available starting with Hive 0.13) |

##### Synopsis

* Directory can be a full URI. If scheme or authority are not specified, Hive will use the scheme and authority from the hadoop configuration variable fs.default.name that specifies the Namenode URI.
* If LOCAL keyword is used, Hive will write data to the directory on the local file system.
* Data written to the filesystem is serialized as text with columns separated by ^A and rows separated by newlines. If any of the columns are not of primitive type, then those columns are serialized to JSON format.

### Inserting values into tables from SQL

The INSERT...VALUES statement can be used to insert data into tables directly from SQL.

**Version Information**

INSERT...VALUES is available starting in [Hive 0.14](https://issues.apache.org/jira/browse/HIVE-5317).

##### Syntax

|  |
| --- |
| Standard Syntax:  INSERT INTO TABLE tablename [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES values\_row [, values\_row ...]    Where values\_row is:  ( value [, value ...] )  where a value is either null or any valid SQL literal |

##### Synopsis

* Each row listed in the VALUES clause is inserted into table tablename.
* Values must be provided for every column in the table. The standard SQL syntax that allows the user to insert values into only some columns is not yet supported. To mimic the standard SQL, nulls can be provided for columns the user does not wish to assign a value to.
* Dynamic partitioning is supported in the same way as for [INSERT...SELECT](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-DynamicPartitionInserts).
* If the table being inserted into supports [ACID](https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions) and a transaction manager that supports ACID is in use, this operation will be auto-committed upon successful completion.
* Hive does not support literals for complex types (array, map, struct, union), so it is not possible to use them in INSERT INTO...VALUES clauses. This means that the user cannot insert data into a complex datatype column using the INSERT INTO...VALUES clause.

##### Examples

|  |
| --- |
| CREATE TABLE students (name VARCHAR(64), age INT, gpa DECIMAL(3, 2))    CLUSTERED BY (age) INTO 2 BUCKETS STORED AS ORC;    INSERT INTO TABLE students    VALUES ('fred flintstone', 35, 1.28), ('barney rubble', 32, 2.32);      CREATE TABLE pageviews (userid VARCHAR(64), link STRING, came\_from STRING)    PARTITIONED BY (datestamp STRING) CLUSTERED BY (userid) INTO 256 BUCKETS STORED AS ORC;    INSERT INTO TABLE pageviews PARTITION (datestamp = '2014-09-23')    VALUES ('jsmith', 'mail.com', 'sports.com'), ('jdoe', 'mail.com', null);    INSERT INTO TABLE pageviews PARTITION (datestamp)    VALUES ('tjohnson', 'sports.com', 'finance.com', '2014-09-23'), ('tlee', 'finance.com', null, '2014-09-21'); |

### Update

**Version Information**

UPDATE is available starting in [Hive 0.14](https://issues.apache.org/jira/browse/HIVE-5317).

Updates can only be performed on tables that support ACID. See [Hive Transactions](https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions) for details.

##### Syntax

|  |
| --- |
| Standard Syntax:  UPDATE tablename SET column = value [, column = value ...] [WHERE expression] |

##### Synopsis

* The referenced column must be a column of the table being updated.
* The value assigned must be an expression that Hive supports in the select clause.  Thus arithmetic operators, UDFs, casts, literals, etc. are supported.  Subqueries are not supported.
* Only rows that match the WHERE clause will be updated.
* Partitioning columns cannot be updated.
* Bucketing columns cannot be updated.
* In Hive 0.14, upon successful completion of this operation the changes will be auto-committed.

##### Notes

* Vectorization will be turned off for update operations.  This is automatic and requires no action on the part of the user.  Non-update operations are not affected.  Updated tables can still be queried using vectorization.
* In version 0.14 it is recommended that you set [hive.optimize.sort.dynamic.partition](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties" \l "ConfigurationProperties-hive.optimize.sort.dynamic.partition)=false when doing updates, as this produces more efficient execution plans.

### Delete

**Version Information**

DELETE is available starting in [Hive 0.14](https://issues.apache.org/jira/browse/HIVE-5317).

Deletes can only be performed on tables that support ACID. See [Hive Transactions](https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions) for details.

##### Syntax

|  |
| --- |
| Standard Syntax:  DELETE FROM tablename [WHERE expression] |

##### Synopsis

* Only rows that match the WHERE clause will be deleted.
* In Hive 0.14, upon successful completion of this operation the changes will be auto-committed.

##### Notes

* Vectorization will be turned off for delete operations.  This is automatic and requires no action on the part of the user.  Non-delete operations are not affected.  Tables with deleted data can still be queried using vectorization.
* In version 0.14 it is recommended that you set [hive.optimize.sort.dynamic.partition](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties" \l "ConfigurationProperties-hive.optimize.sort.dynamic.partition)=false when doing deletes, as this produces more efficient execution plans.

### Merge

**Version Information**

MERGE is available starting in [Hive 2.2](https://issues.apache.org/jira/browse/HIVE-10924).

Merge can only be performed on tables that support ACID. See [Hive Transactions](https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions) for details.

##### Syntax

|  |
| --- |
| Standard Syntax:  MERGE INTO <target table> AS T USING <source expression/table> AS S  ON <boolean expression1>  WHEN MATCHED [AND <boolean expression2>] THEN UPDATE SET <set clause list>  WHEN MATCHED [AND <boolean expression3>] THEN DELETE  WHEN NOT MATCHED [AND <boolean expression4>] THEN INSERT VALUES<value list> |

##### Synopsis

* [Merge](https://en.wikipedia.org/wiki/Merge_(SQL)) allows actions to be performed on a target table based on the results of a join with a source table.
* In Hive 2.2, upon successful completion of this operation the changes will be auto-committed.

##### Performance Note

SQL Standard requires that an error is raised if the ON clause is such that more than 1 row in source matches a row in target.  This check is computationally expensive and may affect the overall runtime of a MERGE statement significantly.  [hive.merge.cardinality.check](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties" \l "ConfigurationProperties-hive.merge.cardinality.check)=false may be used to disable the check at your own risk.  If the check is disabled, but the statement has such a cross join effect, it may lead to data corruption.

##### Notes

* 1, 2, or 3 WHEN clauses may be present; at most 1 of each type:  UPDATE/DELETE/INSERT.
* WHEN NOT MATCHED must be the last WHEN clause.
* If both UPDATE and DELETE clauses are present, the first one in the statement must include [AND <boolean expression>].
* Vectorization will be turned off for merge operations.  This is automatic and requires no action on the part of the user.  Non-delete operations are not affected.  Tables with deleted data can still be queried using vectorization.

**HiveQL Manipulations**

## What is HiveQL(Hive Query Language)?

Hive provides a CLI to write Hive queries using Hive Query Language (HiveQL). Generally HQL syntax is similar to the[SQL](https://www.guru99.com/sql.html)syntax that most data analysts are familiar with.

Hive's SQL-inspired language separates the user from the complexity of Map Reduce programming. It reuses familiar concepts from the relational database world, such as tables, rows, columns and schema, to ease learning.

Most interactions tend to take place over a command line interface (CLI). Hive provides a CLI to write Hive queries using Hive Query Language (Hive-QL).

Generally, HiveQL syntax is similar to the[SQL](https://www.guru99.com/sql.html)syntax that most data analysts are familiar with. Hive supports four file formats those are TEXTFILE, SEQUENCEFILE, ORC and RCFILE (Record Columnar File).

* For single user metadata storage Hive uses derby database and
* For multiple user Metadata or shared Metadata case Hive uses MYSQL

## Built-in operators

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

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

Types of Built-in Operators in HIVE are:

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

**Relational Operators:**

We use Relational operators for relationship comparisons between two operands.

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

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

|  |  |  |
| --- | --- | --- |
| **Built-in Operator** | **Description** | **Operand** |
| X = Y | TRUE   if expression X is equivalent to expression Y   Otherwise FALSE. | It takes all primitive types |
| X != Y | TRUE  if expression X is not equivalent to expression Y  Otherwise FALSE. | It takes all primitive types |
| X < Y | TRUE  if expression X is less than expression Y   Otherwise FALSE. | It takes all primitive types |
| X <= Y | TRUE   if expression X is less than or equal to expression Y   Otherwise FALSE. | It takes all primitive types |
| X>Y | TRUE  if expression X is greater than expression Y  Otherwise FALSE. | It takes all primitive types |
| X>= Y | TRUE   if expression X is greater than or equal to expression Y  Otherwise FALSE. | It takes all primitive types |
| X IS NULL | TRUE if expression X evaluates to NULL otherwise FALSE. | It takes all types |
| X IS NOT NULL | FALSE   If expression X evaluates to NULL otherwise TRUE. | It takes all types |
| X LIKE Y | TRUE   If string pattern X matches to Y otherwise FALSE. | Takes only Strings |
| X RLIKE Y | NULL if X or Y is NULL, TRUE if any substring of X matches the[Java](https://www.guru99.com/java-tutorial.html)regular expression Y, otherwise FALSE. | Takes only Strings |
| X REGEXP Y | Same as RLIKE. | Takes only Strings |

**Arithmetic Operators**:

We use Arithmetic operators for performing arithmetic operations on operands

* Arithmetic operations such as addition, subtraction, multiplication and division between operands we use these Operators.
* The operand types all are number types in these Operators

**Sample Example:**

**2 + 3 gives result 5.**

In this example, '+' is theoperator and 2 and 3 are operands. The return value is 5

The following Table will give us details about Arithmetic operators

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

**Logical Operators:**

We use Logical operators for performing Logical operations on operands

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

The following Table will give us details about Logical operators

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

**Operators on Complex types:**

The following Table will give us details about Complex Type Operators . These are operators which will provide a different mechanism to access elements in complex types.

|  |  |  |
| --- | --- | --- |
| **Operators** | **Operands** | **Description** |
| A[n] | A is an Array and n is an integer type | It will return nth element in the array A. The first element has index of 0 |
| M[key] | M is a Map<K, V> and key has type K | It will return the values belongs to the key in the map |

**Complex type Constructors:**

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

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

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

**Summary:**

Hive provides some inbuilt functions and operators to manipulate the data stored in Hive warehouse. Hive is similar to[SQL](https://www.guru99.com/sql.html)language, which supports all type of data operations and querying on tables and databases.