

# LanguageManual DML

## Hive Data Manipulation Language

- [Hive Data Manipulation Language](#)
  - [Loading files into tables](#)
    - [Syntax](#)
    - [Synopsis](#)
    - [Notes](#)
  - [Inserting data into Hive Tables from queries](#)
    - [Syntax](#)
    - [Synopsis](#)
    - [Notes](#)
    - [Dynamic Partition Inserts](#)
      - [Example](#)
      - [Additional Documentation](#)
  - [Writing data into the filesystem from queries](#)
    - [Syntax](#)
    - [Synopsis](#)
    - [Notes](#)
  - [Inserting values into tables from SQL](#)
    - [Syntax](#)
    - [Synopsis](#)
    - [Examples](#)
  - [Update](#)
    - [Syntax](#)
    - [Synopsis](#)
    - [Notes](#)
  - [Delete](#)
    - [Syntax](#)
    - [Synopsis](#)
    - [Notes](#)
  - [Merge](#)
    - [Syntax](#)
    - [Synopsis](#)
    - [Performance Note](#)
    - [Notes](#)
    - [Examples](#)

There are multiple ways to modify data in Hive:

- [LOAD](#)
- [INSERT](#)
  - [into Hive tables from queries](#)
  - [into directories from queries](#)
  - [into Hive tables from SQL](#)
- [UPDATE](#)
- [DELETE](#)
- [MERGE](#)

[EXPORT](#) and [IMPORT](#) commands are also available (as of [Hive 0.8](#)).

### 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 ...)]

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)] [INPUTFORMAT 'inputformat' SERDE 'serde'] (3.0 or later)
```

#### Synopsis

Load operations prior to Hive 3.0 are 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`
  - 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.
  - Note: If you run this command against a HiveServer2 instance then the local path refers to a path on the HiveServer2 instance. HiveServer2 must have the proper permissions to access that file.
- 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.name` that 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.

Additional load operations are supported by Hive 3.0 onwards as Hive internally rewrites the load into an INSERT AS SELECT.

- If table has partitions, however, the load command does not have them, the load would be converted into INSERT AS SELECT and assume that the last set of columns are partition columns. It will throw an error if the file does not conform to the expected schema.
- If table is bucketed then the following rules apply:
  - In strict mode : launches an INSERT AS SELECT job.
  - In non-strict mode : if the file names conform to the naming convention (if the file belongs to bucket 0, it should be named 000000\_0 or 000000\_0\_copy\_1, or if it belongs to bucket 2 the names should be like 000002\_0 or 000002\_0\_copy\_3, etc.) then it will be a pure copy /move operation, else it will launch an INSERT AS SELECT job.
- *filepath* can contain subdirectories, provided each file conforms to the schema.
- *inputformat* can be any Hive input format such as text, ORC, etc.
- *serde* can be the associated Hive SERDE.
- Both *inputformat* and *serde* are case sensitive.

Example of such a schema:

```
CREATE TABLE tabl (col1 int, col2 int) PARTITIONED BY (col3 int) STORED AS ORC;
LOAD DATA LOCAL INPATH 'filepath' INTO TABLE tabl;
```

Here, partition information is missing which would otherwise give an error, however, if the file(s) located at *filepath* conform to the table schema such that each row ends with partition column(s) then the load will rewrite into an INSERT AS SELECT job.

The uncompressed data should look like this:

(1,2,3), (2,3,4), (4,5,3) etc.

## Notes

- *filepath* cannot contain subdirectories (except for Hive 3.0 or later, as described above).
- 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](#)).
- Please read [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](#)).
  - As of Hive 2.3.0 ([HIVE-15880](#)), if the table has [TBLPROPERTIES](#) ("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](#)) 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](#), a table can be made *immutable* by creating it with [TBLPROPERTIES](#) ("immutable"="true"). 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](#) is set to true, these values are validated, converted and normalized to conform to their column types (Hive [0.12.0](#) 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](#), if a table has an [OutputFormat](#) that implements [AcidOutputFormat](#) and the system is configured to use a [transaction](#) 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](#) (for non-partitioned tables) or [DROP PARTITION](#) followed by INSERT INTO.
- As of Hive [1.1.0](#) the TABLE keyword is optional.
- As of Hive [1.2.0](#) each INSERT INTO T can take a column list like INSERT INTO T (z, x, c1). See Description of [HIVE-9481](#) for examples.
- As of Hive [3.1.0](#) INSERT OVERWRITE from a source with UNION ALL on full CRUD ACID tables is not allowed.

## 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](#), 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](#).

## 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. As of Hive 3.0.0 ([HIVE-19083](#)) there is no need to specify dynamic partition columns. Hive will automatically generate partition specification if it is not specified.

Dynamic partition inserts are disabled by default prior to Hive 0.9.0 and enabled by default in Hive [0.9.0](#) and later. These are the relevant configuration properties for dynamic partition inserts:

Configuration property	Default	Note
<code>hive.exec.dynamic.partition</code>	<code>true</code>	Needs to be set to <code>true</code> to enable dynamic partition inserts
<code>hive.exec.dynamic.partition.mode</code>	<code>strict</code>	In <code>strict</code> mode, the user must specify at least one static partition in case the user accidentally overwrites all partitions, in <code>nonstrict</code> mode all partitions are allowed to be dynamic
<code>hive.exec.max.dynamic.partitions.pernode</code>	100	Maximum number of dynamic partitions allowed to be created in each mapper/reducer node
<code>hive.exec.max.dynamic.partitions</code>	1000	Maximum number of dynamic partitions allowed to be created in total
<code>hive.exec.max.created.files</code>	100000	Maximum number of HDFS files created by all mappers/reducers in a MapReduce job
<code>hive.error.on.empty.partition</code>	<code>false</code>	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](#)
  - [Original design doc](#)
  - [HIVE-936](#)
- [Tutorial: Dynamic-Partition Insert](#)
- [HCatalog Dynamic Partitioning](#)
  - [Usage with Pig](#)
  - [Usage from MapReduce](#)

## 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.

## Notes

- `INSERT OVERWRITE` statements to directories, local directories, and tables (or partitions) can all be used together within the same query.
- `INSERT OVERWRITE` statements to HDFS filesystem directories are the best way to extract large amounts of data from Hive. Hive can write to HDFS directories in parallel from within a map-reduce job.
- The directory is, as you would expect, `OVERWRIT`ten; in other words, if the specified path exists, it is clobbered and replaced with the output.
- As of Hive 0.11.0 the separator used can be specified; in earlier versions it was always the `^A` character (`\001`). However, custom separators are only supported for `LOCAL` writes in Hive versions 0.11.0 to 1.1.0 – this bug is fixed in version 1.2.0 (see [HIVE-5672](#)).
- In [Hive 0.14](#), inserts into `ACID` compliant tables will deactivate vectorization for the duration of the select and insert. This will be done automatically. `ACID` tables that have data inserted into them can still be queried using vectorization.

## 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](#).

## 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](#).
- If the table being inserted into supports `ACID` 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');
```

```
INSERT INTO TABLE pageviews
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](#).

Updates can only be performed on tables that support ACID. See [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=false](#) when doing updates, as this produces more efficient execution plans.

## Delete



### Version Information

DELETE is available starting in [Hive 0.14](#).

Deletes can only be performed on tables that support ACID. See [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=false](#) when doing deletes, as this produces more efficient execution plans.

## Merge



### Version Information

MERGE is available starting in [Hive 2.2](#).

Merge can only be performed on tables that support ACID. See [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](#) 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=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.

### Examples

- See [here](#).