15.2.7 INSERT Statement


```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    { {VALUES | VALUE} (value_list) [, (value_list)] ... }
    [AS row_alias[(col_alias [, col_alias] ...)]]
    [ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET assignment_list
    [AS row_alias[(col_alias [, col_alias] ...)]]
    [ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    { SELECT ...
      | TABLE table_name
      | VALUES row_constructor_list
    }
    [ON DUPLICATE KEY UPDATE assignment_list]

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...

row_constructor_list:
    ROW(value_list)[, ROW(value_list)][, ...]

assignment:
    col_name =
          value
        | [row_alias.]col_name
        | [tbl_name.]col_name
        | [row_alias.]col_alias
```

```
assignment_list:
    assignment [, assignment] ...
```

INSERT inserts new rows into an existing table. The INSERT ... VALUES, INSERT ... VALUES ROW(), and INSERT ... SET forms of the statement insert rows based on explicitly specified values. The INSERT ... SELECT form inserts rows selected from another table or tables. You can also use INSERT ... TABLE in MySQL 8.0.19 and later to insert rows from a single table. INSERT with an ON DUPLICATE KEY UPDATE clause enables existing rows to be updated if a row to be inserted would cause a duplicate value in a UNIQUE index or PRIMARY KEY. In MySQL 8.0.19 and later, a row alias with one or more optional column aliases can be used with ON DUPLICATE KEY UPDATE to refer to the row to be inserted.

For additional information about INSERT ... SELECT and INSERT ... ON DUPLICATE KEY UPDATE, see Section 15.2.7.1, “INSERT ... SELECT Statement”, and Section 15.2.7.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”.

In MySQL 8.0, the DELAYED keyword is accepted but ignored by the server. For the reasons for this, see Section 15.2.7.3, “INSERT DELAYED Statement”,

Inserting into a table requires the INSERT privilege for the table. If the ON DUPLICATE KEY UPDATE clause is used and a duplicate key causes an UPDATE to be performed instead, the statement requires the UPDATE privilege for the columns to be updated. For columns that are read but not modified you need only the SELECT privilege (such as for a column referenced only on the right hand side of an ***col_name*=*expr*** assignment in an ON DUPLICATE KEY UPDATE clause).

When inserting into a partitioned table, you can control which partitions and subpartitions accept new rows. The PARTITION clause takes a list of the comma-separated names of one or more partitions or subpartitions (or both) of the table. If any of the rows to be inserted by a given INSERT statement do not match one of the partitions listed, the INSERT statement fails with the error Found a row not matching the given partition set. For more information and examples, see Section 26.5, “Partition Selection”.

tbl_name is the table into which rows should be inserted. Specify the columns for which the statement provides values as follows:

- Provide a parenthesized list of comma-separated column names following the table name. In this case, a value for each named column must be provided by the VALUES list, VALUES ROW() list, or SELECT statement. For the INSERT TABLE form, the number of columns in the source table must match the number of columns to be inserted.
- If you do not specify a list of column names for INSERT ... VALUES or INSERT ... SELECT, values for every column in the table must be provided by the VALUES list, SELECT statement, or

TABLE statement. If you do not know the order of the columns in the table, use DESCRIBE *tbl_name* to find out.

- A SET clause indicates columns explicitly by name, together with the value to assign each one.

Column values can be given in several ways:

- If strict SQL mode is not enabled, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in Section 13.6, “Data Type Default Values”. See also Section 1.6.3.3, “Enforced Constraints on Invalid Data”.

  If strict SQL mode is enabled, an INSERT statement generates an error if it does not specify an explicit value for every column that has no default value. See Section 7.1.11, “Server SQL Modes”.

- If both the column list and the VALUES list are empty, INSERT creates a row with each column set to its default value:

  ```
  INSERT INTO tbl_name () VALUES();
  ```

  If strict mode is not enabled, MySQL uses the implicit default value for any column that has no explicitly defined default. If strict mode is enabled, an error occurs if any column has no default value.

- Use the keyword DEFAULT to set a column explicitly to its default value. This makes it easier to write INSERT statements that assign values to all but a few columns, because it enables you to avoid writing an incomplete VALUES list that does not include a value for each column in the table. Otherwise, you must provide the list of column names corresponding to each value in the VALUES list.

- If a generated column is inserted into explicitly, the only permitted value is DEFAULT. For information about generated columns, see Section 15.1.20.8, “CREATE TABLE and Generated Columns”.

- In expressions, you can use DEFAULT(*col_name*) to produce the default value for column *col_name*.

- Type conversion of an expression *expr* that provides a column value might occur if the expression data type does not match the column data type. Conversion of a given value can result in different inserted values depending on the column type. For example, inserting the string '1999.0e-2' into an INT, FLOAT, DECIMAL(10,6), or YEAR column inserts the value 1999, 19.9921, 19.992100, or 1999, respectively. The value stored in the INT and YEAR columns is 1999 because the string-to-number conversion looks only at as much of the initial part of the string as may be

considered a valid integer or year. For the `FLOAT` and `DECIMAL` columns, the string-to-number conversion considers the entire string a valid numeric value.

- An expression ***expr*** can refer to any column that was set earlier in a value list. For example, you can do this because the value for `col2` refers to `col1`, which has previously been assigned:

  ```
  INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  But the following is not legal, because the value for `col1` refers to `col2`, which is assigned after `col1`:

  ```
  INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```

  An exception occurs for columns that contain `AUTO_INCREMENT` values. Because `AUTO_INCREMENT` values are generated after other value assignments, any reference to an `AUTO_INCREMENT` column in the assignment returns a `0`.

`INSERT` statements that use `VALUES` syntax can insert multiple rows. To do this, include multiple lists of comma-separated column values, with lists enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c)
    VALUES(1,2,3), (4,5,6), (7,8,9);
```

Each values list must contain exactly as many values as are to be inserted per row. The following statement is invalid because it contains one list of nine values, rather than three lists of three values each:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

`VALUE` is a synonym for `VALUES` in this context. Neither implies anything about the number of values lists, nor about the number of values per list. Either may be used whether there is a single values list or multiple lists, and regardless of the number of values per list.

`INSERT` statements using `VALUES ROW()` syntax can also insert multiple rows. In this case, each value list must be contained within a `ROW()` (row constructor), like this:

```
INSERT INTO tbl_name (a,b,c)
    VALUES ROW(1,2,3), ROW(4,5,6), ROW(7,8,9);
```

The affected-rows value for an `INSERT` can be obtained using the `ROW_COUNT()` SQL function or the `mysql_affected_rows()` C API function. See Section 14.15, “Information Functions”, and mysql_affected_rows().

If you use `INSERT ... VALUES` or `INSERT ... VALUES ROW()` with multiple value lists, or `INSERT ... SELECT` or `INSERT ... TABLE`, the statement returns an information string in this format:

```
Records: N1 Duplicates: N2 Warnings: N3
```

If you are using the C API, the information string can be obtained by invoking the `mysql_info()` function. See mysql_info().

`Records` indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because `Duplicates` can be nonzero.) `Duplicates` indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. For multiple-row `INSERT` statements or `INSERT INTO ... SELECT` statements, the column is set to the implicit default value for the column data type. This is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. `INSERT INTO ... SELECT` statements are handled the same way as multiple-row inserts because the server does not examine the result set from the `SELECT` to see whether it returns a single row. (For a single-row `INSERT`, no warning occurs when `NULL` is inserted into a `NOT NULL` column. Instead, the statement fails with an error.)

- Setting a numeric column to a value that lies outside the column range. The value is clipped to the closest endpoint of the range.

- Assigning a value such as `'10.34 a'` to a numeric column. The trailing nonnumeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to `0`.

- Inserting a string into a string column (`CHAR`, `VARCHAR`, `TEXT`, or `BLOB`) that exceeds the column maximum length. The value is truncated to the column maximum length.

- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

- For `INSERT` examples involving `AUTO_INCREMENT` column values, see Section 5.6.9, “Using AUTO_INCREMENT”.

If `INSERT` inserts a row into a table that has an `AUTO_INCREMENT` column, you can find the value used for that column by using the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function.

> **Note**
>
> These two functions do not always behave identically. The behavior of `INSERT` statements with respect to `AUTO_INCREMENT` columns is discussed further in Section 14.15, “Information Functions”, and mysql_insert_id().

The `INSERT` statement supports the following modifiers:

- If you use the `LOW_PRIORITY` modifier, execution of the `INSERT` is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the `INSERT LOW_PRIORITY` statement is waiting. It is possible, therefore, for a client that issues an `INSERT LOW_PRIORITY` statement to wait for a very long time.

  `LOW_PRIORITY` affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

  > **Note**
  >
  > `LOW_PRIORITY` should normally not be used with `MyISAM` tables because doing so disables concurrent inserts. See Section 10.11.3, “Concurrent Inserts”.

- If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used. See Section 10.11.3, “Concurrent Inserts”.

  `HIGH_PRIORITY` affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- If you use the `IGNORE` modifier, ignorable errors that occur while executing the `INSERT` statement are ignored. For example, without `IGNORE`, a row that duplicates an existing `UNIQUE` index or `PRIMARY KEY` value in the table causes a duplicate-key error and the statement is aborted. With `IGNORE`, the row is discarded and no error occurs. Ignored errors generate warnings instead.

  `IGNORE` has a similar effect on inserts into partitioned tables where no partition matching a given value is found. Without `IGNORE`, such `INSERT` statements are aborted with an error. When `INSERT IGNORE` is used, the insert operation fails silently for rows containing the unmatched value, but inserts rows that are matched. For an example, see Section 26.2.2, “LIST Partitioning”.

Data conversions that would trigger errors abort the statement if `IGNORE` is not specified. With `IGNORE`, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the `mysql_info()` C API function how many rows were actually inserted into the table.

For more information, see The Effect of IGNORE on Statement Execution.

You can use `REPLACE` instead of `INSERT` to overwrite old rows. `REPLACE` is the counterpart to `INSERT IGNORE` in the treatment of new rows that contain unique key values that duplicate old rows: The new rows replace the old rows rather than being discarded. See Section 15.2.12, “REPLACE Statement”.

- If you specify `ON DUPLICATE KEY UPDATE`, and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row occurs. The affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the `CLIENT_FOUND_ROWS` flag to the `mysql_real_connect()` C API function when connecting to **mysqld**, the affected-rows value is 1 (not 0) if an existing row is set to its current values. See Section 15.2.7.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”.

- `INSERT DELAYED` was deprecated in MySQL 5.6, and is scheduled for eventual removal. In MySQL 8.0, the `DELAYED` modifier is accepted but ignored. Use `INSERT` (without `DELAYED`) instead. See Section 15.2.7.3, “INSERT DELAYED Statement”.