

### 13.1.1. DELETE Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
      [WHERE where_definition]
      [ORDER BY ...]
      [LIMIT row_count]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      tbl_name[.*] [, tbl_name[.*] ...]
FROM table_references
      [WHERE where_definition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      FROM tbl_name[.*] [, tbl_name[.*] ...]
      USING table_references
      [WHERE where_definition]
```

DELETE deletes rows from *tbl\_name* that satisfy the condition given by *where\_definition*, and returns the number of records deleted.

If you issue a DELETE statement with no WHERE clause, all rows are deleted. A faster way to do this, when you don't want to know the number of deleted rows, is to use TRUNCATE TABLE. See [Section 13.1.9, "TRUNCATE Syntax"](#).

In MySQL 3.23, DELETE without a WHERE clause returns zero as the number of affected records.

In MySQL 3.23, if you really want to know how many records are deleted when you are deleting all rows, and are willing to suffer a speed penalty, you can use a DELETE statement that includes a WHERE clause with an expression that is true for every row. For example:

```
mysql> DELETE FROM tbl_name WHERE 1>0;
```

This is much slower than TRUNCATE *tbl\_name*, because it deletes rows one at a time.

If you delete the row containing the maximum value for an AUTO\_INCREMENT column, the value is reused for an ISAM or BDB table, but not for a MyISAM or InnoDB table. If you delete all rows in the table with DELETE FROM *tbl\_name* (without a WHERE) in AUTOCOMMIT mode, the sequence starts over for all table types except for InnoDB and (as of MySQL 4.0) MyISAM. There are some exceptions to this behavior for InnoDB tables, discussed in [Section 15.7.3, "How an AUTO\\_INCREMENT Column Works in InnoDB"](#).

For MyISAM and BDB tables, you can specify an AUTO\_INCREMENT secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for MyISAM tables. See [Section 3.6.9, "Using AUTO\\_INCREMENT"](#).

The `DELETE` statement supports the following modifiers:

- If you specify the `LOW_PRIORITY` keyword, execution of the `DELETE` is delayed until no other clients are reading from the table.
- For `MyISAM` tables, if you specify the `QUICK` keyword, the storage engine does not merge index leaves during delete, which may speed up certain kind of deletes.
- The `IGNORE` keyword causes MySQL to ignore all errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of this option are returned as warnings. This option first appeared in MySQL 4.1.1.

The speed of delete operations may also be affected by factors discussed in [Section 7.2.16, “Speed of `DELETE` Statements”](#).

In `MyISAM` tables, deleted records are maintained in a linked list and subsequent `INSERT` operations reuse old record positions. To reclaim unused space and reduce file sizes, use the `OPTIMIZE TABLE` statement or the `myisamchk` utility to reorganize tables. `OPTIMIZE TABLE` is easier, but `myisamchk` is faster. See [Section 13.5.2.5, “`OPTIMIZE TABLE` Syntax”](#) and [Section 5.8.3.10, “Table Optimization”](#).

The `QUICK` modifier affects whether index leaves are merged for delete operations. `DELETE QUICK` is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

`DELETE QUICK` is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of `QUICK` can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed `AUTO_INCREMENT` column.
2. Insert many records into the table. Each insert results in an index values that is added to the high end of the index.
3. Delete a block of records at the low end of the column range using `DELETE QUICK`.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of `QUICK`. They remain underfilled when new inserts occur, because new records does not have index values in the deleted range. Furthermore, they remain underfilled even if you later use `DELETE` without `QUICK`, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, you can use `OPTIMIZE TABLE`.

If you are going to delete many rows from a table, it might be faster to use `DELETE QUICK` followed by `OPTIMIZE TABLE`. This rebuilds the index rather than performing many index block merge operations.

The MySQL-specific `LIMIT row_count` option to `DELETE` tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to

ensure that a specific `DELETE` statement doesn't take too much time. You can simply repeat the `DELETE` statement until the number of affected rows is less than the `LIMIT` value.

If the `DELETE` statement includes an `ORDER BY` clause, the rows are deleted in the order specified by the clause. This is really useful only in conjunction with `LIMIT`. For example, the following statement finds rows matching the `WHERE` clause, sorts them in `timestamp` order, and deletes the first (oldest) one:

```
DELETE FROM someLog
WHERE user = 'jcole'
ORDER BY timestamp
LIMIT 1
```

`ORDER BY` can be used with `DELETE` beginning with MySQL 4.0.0.

From MySQL 4.0, you can specify multiple tables in the `DELETE` statement to delete rows from one or more tables depending on a particular condition in multiple tables. However, you cannot use `ORDER BY` or `LIMIT` in a multiple-table `DELETE`.

The first multiple-table `DELETE` syntax is supported starting from MySQL 4.0.0. The second is supported starting from MySQL 4.0.2. The *table\_references* part lists the tables involved in the join. Its syntax is described in [Section 13.1.7.1, "JOIN Syntax"](#).

For the first syntax, only matching rows from the tables listed before the `FROM` clause are deleted. For the second syntax, only matching rows from the tables listed in the `FROM` clause (before the `USING` clause) are deleted. The effect is that you can delete rows from many tables at the same time and also have additional tables that are used for searching:

```
DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three files when searching for rows to delete, but delete matching rows only from tables `t1` and `t2`.

The examples show inner joins using the comma operator, but multiple-table `DELETE` statements can use any type of join allowed in `SELECT` statements, such as `LEFT JOIN`.

The syntax allows `. *` after the table names for compatibility with *Access*.

If you use a multiple-table `DELETE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, delete from a single table and rely on the `ON DELETE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly.

**Note:** In MySQL 4.0, you should refer to the table names to be deleted with the true table name. In MySQL 4.1, you must use the alias (if one was given) when referring to a table name:

In MySQL 4.0:

```
DELETE test FROM test AS t1, test2 WHERE ...
```

In MySQL 4.1:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

The reason we didn't make this change in 4.0 was to avoid breaking any old 4.0 applications that were using the old syntax.

Cross-database deletes are supported for multiple-table deletes, but in this case, you must refer to the tables without using aliases. For example:

```
DELETE test1.tmp1, test2.tmp2 FROM test1.tmp1, test2.tmp2 WHERE ...
```

Currently, you cannot delete from a table and select from the same table in a subquery.