13.1.9 ALTER TABLE Statement


```
ALTER TABLE tbl_name
    [alter_option [, alter_option] ...]
    [partition_options]

alter_option: {
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX | KEY} [index_name]
        [index_type] (key_part,...) [index_option] ...
  | ADD {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name]
        (key_part,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (key_part,...)
        [index_option] ...
  | ADD [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
        [index_name] [index_type] (key_part,...)
        [index_option] ...
  | ADD [CONSTRAINT [symbol]] FOREIGN KEY
        [index_name] (col_name,...)
        reference_definition
  | ADD [CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]
  | DROP {CHECK | CONSTRAINT} symbol
  | ALTER {CHECK | CONSTRAINT} symbol [NOT] ENFORCED
  | ALGORITHM [=] {DEFAULT | INSTANT | INPLACE | COPY}
  | ALTER [COLUMN] col_name {
        SET DEFAULT {literal | (expr)}
      | SET {VISIBLE | INVISIBLE}
      | DROP DEFAULT
    }
  | ALTER INDEX index_name {VISIBLE | INVISIBLE}
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST | AFTER col_name]
  | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | {DISABLE | ENABLE} KEYS
  | {DISCARD | IMPORT} TABLESPACE
```

```
  | DROP [COLUMN] col_name
  | DROP {INDEX | KEY} index_name
  | DROP PRIMARY KEY
  | DROP FOREIGN KEY fk_symbol
  | FORCE
  | LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ORDER BY col_name [, col_name] ...
  | RENAME COLUMN old_col_name TO new_col_name
  | RENAME {INDEX | KEY} old_index_name TO new_index_name
  | RENAME [TO | AS] new_tbl_name
  | {WITHOUT | WITH} VALIDATION
}

partition_options:
    partition_option [partition_option] ...

partition_option: {
    ADD PARTITION (partition_definition)
  | DROP PARTITION partition_names
  | DISCARD PARTITION {partition_names | ALL} TABLESPACE
  | IMPORT PARTITION {partition_names | ALL} TABLESPACE
  | TRUNCATE PARTITION {partition_names | ALL}
  | COALESCE PARTITION number
  | REORGANIZE PARTITION partition_names INTO (partition_definitions)
  | EXCHANGE PARTITION partition_name WITH TABLE tbl_name [{WITH | WITHOUT} VALIDATI
  | ANALYZE PARTITION {partition_names | ALL}
  | CHECK PARTITION {partition_names | ALL}
  | OPTIMIZE PARTITION {partition_names | ALL}
  | REBUILD PARTITION {partition_names | ALL}
  | REPAIR PARTITION {partition_names | ALL}
  | REMOVE PARTITIONING
}

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}
}

table_options:
```

```
    table_option [[,] table_option] ...

table_option: {
    AUTOEXTEND_SIZE [=] value
  | AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | COMPRESSION [=] {'ZLIB' | 'LZ4' | 'NONE'}
  | CONNECTION [=] 'connect_string'
  | {DATA | INDEX} DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | ENCRYPTION [=] {'Y' | 'N'}
  | ENGINE [=] engine_name
  | ENGINE_ATTRIBUTE [=] 'string'
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT}
  | SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
  | STATS_AUTO_RECALC [=] {DEFAULT | 0 | 1}
  | STATS_PERSISTENT [=] {DEFAULT | 0 | 1}
  | STATS_SAMPLE_PAGES [=] value
  | TABLESPACE tablespace_name [STORAGE {DISK | MEMORY}]
  | UNION [=] (tbl_name[,tbl_name]...)
}

partition_options:
    (see CREATE TABLE options)
```

`ALTER TABLE` changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change characteristics such as the storage engine used for the table or the table comment.

- To use `ALTER TABLE`, you need `ALTER`, `CREATE`, and `INSERT` privileges for the table. Renaming a table requires `ALTER` and `DROP` on the old table, `ALTER`, `CREATE`, and `INSERT` on the new table.
- Following the table name, specify the alterations to be made. If none are given, `ALTER TABLE` does nothing.
- The syntax for many of the permissible alterations is similar to clauses of the `CREATE TABLE` statement. ***`column_definition`*** clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE`

TABLE. For more information, see Section 13.1.20, “CREATE TABLE Statement”.

- The word COLUMN is optional and can be omitted, except for RENAME COLUMN (to distinguish a column-renaming operation from the RENAME table-renaming operation).

- Multiple ADD, ALTER, DROP, and CHANGE clauses are permitted in a single ALTER TABLE statement, separated by commas. This is a MySQL extension to standard SQL, which permits only one of each clause per ALTER TABLE statement. For example, to drop multiple columns in a single statement, do this:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- If a storage engine does not support an attempted ALTER TABLE operation, a warning may result. Such warnings can be displayed with SHOW WARNINGS. See Section 13.7.7.42, “SHOW WARNINGS Statement”. For information on troubleshooting ALTER TABLE, see Section B.3.6.1, “Problems with ALTER TABLE”.

- For information about generated columns, see Section 13.1.9.2, “ALTER TABLE and Generated Columns”.

- For usage examples, see Section 13.1.9.3, “ALTER TABLE Examples”.

- InnoDB in MySQL 8.0.17 and later supports addition of multi-valued indexes on JSON columns using a ***key_part*** specification can take the form (CAST ***json_path*** AS ***type*** ARRAY). See Multi-Valued Indexes, for detailed information regarding multi-valued index creation and usage of, as well as restrictions and limitations on multi-valued indexes.

- With the mysql_info() C API function, you can find out how many rows were copied by ALTER TABLE. See mysql_info().

There are several additional aspects to the ALTER TABLE statement, described under the following topics in this section:

- Table Options

- Performance and Space Requirements

- Concurrency Control

- Adding and Dropping Columns

- Renaming, Redefining, and Reordering Columns

- Primary Keys and Indexes
- Foreign Keys and Other Constraints
- Changing the Character Set
- Importing InnoDB Tables
- Row Order for MyISAM Tables
- Partitioning Options

Table Options

table_options signifies table options of the kind that can be used in the `CREATE TABLE` statement, such as `ENGINE`, `AUTO_INCREMENT`, `AVG_ROW_LENGTH`, `MAX_ROWS`, `ROW_FORMAT`, or `TABLESPACE`.

For descriptions of all table options, see Section 13.1.20, “CREATE TABLE Statement”. However, `ALTER TABLE` ignores `DATA DIRECTORY` and `INDEX DIRECTORY` when given as table options. `ALTER TABLE` permits them only as partitioning options, and requires that you have the `FILE` privilege.

Use of table options with `ALTER TABLE` provides a convenient way of altering single table characteristics. For example:

- If `t1` is currently not an `InnoDB` table, this statement changes its storage engine to `InnoDB`:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

  - See Section 15.6.1.5, “Converting Tables from MyISAM to InnoDB” for considerations when switching tables to the `InnoDB` storage engine.
  - When you specify an `ENGINE` clause, `ALTER TABLE` rebuilds the table. This is true even if the table already has the specified storage engine.
  - Running `ALTER TABLE tbl_name ENGINE=INNODB` on an existing `InnoDB` table performs a “null” `ALTER TABLE` operation, which can be used to defragment an `InnoDB` table, as described in Section 15.11.4, “Defragmenting a Table”. Running `ALTER TABLE tbl_name FORCE` on an `InnoDB` table performs the same function.
  - `ALTER TABLE tbl_name ENGINE=INNODB` and `ALTER TABLE tbl_name FORCE` use online DDL. For more information, see Section 15.12, “InnoDB and Online DDL”.

  - The outcome of attempting to change the storage engine of a table is affected by whether the desired storage engine is available and the setting of the `NO_ENGINE_SUBSTITUTION` SQL mode, as described in Section 5.1.11, “Server SQL Modes”.

  - To prevent inadvertent loss of data, `ALTER TABLE` cannot be used to change the storage engine of a table to `MERGE` or `BLACKHOLE`.

- To change the `InnoDB` table to use compressed row-storage format:

```
ALTER TABLE t1 ROW_FORMAT = COMPRESSED;
```

- The `ENCRYPTION` clause enables or disables page-level data encryption for an `InnoDB` table. A keyring plugin must be installed and configured to enable encryption.

  If the `table_encryption_privilege_check` variable is enabled, the `TABLE_ENCRYPTION_ADMIN` privilege is required to use an `ENCRYPTION` clause with a setting that differs from the default schema encryption setting.

  Prior to MySQL 8.0.16, the `ENCRYPTION` clause was only supported when altering tables residing in file-per-table tablespaces. As of MySQL 8.0.16, the `ENCRYPTION` clause is also supported for tables residing in general tablespaces.

  For tables that reside in general tablespaces, table and tablespace encryption must match.

  Altering table encryption by moving a table to a different tablespace or changing the storage engine is not permitted without explicitly specifying an `ENCRYPTION` clause.

  As of MySQL 8.0.16, specifying an `ENCRYPTION` clause with a value other than `'N'` or `''` is not permitted if the table uses a storage engine that does not support encryption. Previously, the clause was accepted. Attempting to create a table without an `ENCRYPTION` clause in an encryption-enabled schema using a storage engine that does not support encryption is also not permitted.

  For more information, see Section 15.13, “InnoDB Data-at-Rest Encryption”.

- To reset the current auto-increment value:

```
ALTER TABLE t1 AUTO_INCREMENT = 13;
```

  You cannot reset the counter to a value less than or equal to the value that is currently in use. For both `InnoDB` and `MyISAM`, if the value is less than or equal to the maximum value currently in the

AUTO_INCREMENT column, the value is reset to the current maximum AUTO_INCREMENT column value plus one.

- To change the default table character set:

```
ALTER TABLE t1 CHARACTER SET = utf8mb4;
```

  See also Changing the Character Set.

- To add (or change) a table comment:

```
ALTER TABLE t1 COMMENT = 'New table comment';
```

- Use ALTER TABLE with the TABLESPACE option to move InnoDB tables between existing general tablespaces, file-per-table tablespaces, and the system tablespace. See Moving Tables Between Tablespaces Using ALTER TABLE.

  - ALTER TABLE ... TABLESPACE operations always cause a full table rebuild, even if the TABLESPACE attribute has not changed from its previous value.

  - ALTER TABLE ... TABLESPACE syntax does not support moving a table from a temporary tablespace to a persistent tablespace.

  - The DATA DIRECTORY clause, which is supported with CREATE TABLE ... TABLESPACE, is not supported with ALTER TABLE ... TABLESPACE, and is ignored if specified.

  - For more information about the capabilities and limitations of the TABLESPACE option, see CREATE TABLE.

- MySQL NDB Cluster 8.0 supports setting NDB_TABLE options for controlling a table's partition balance (fragment count type), read-from-any-replica capability, full replication, or any combination of these, as part of the table comment for an ALTER TABLE statement in the same manner as for CREATE TABLE, as shown in this example:

```
ALTER TABLE t1 COMMENT = "NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RA_BY_NOD
```

  It is also possible to set NDB_COMMENT options for columns of NDB tables as part of an ALTER TABLE statement, like this one:

```
ALTER TABLE t1
  CHANGE COLUMN c1 c1 BLOB
    COMMENT = 'NDB_COLUMN=BLOB_INLINE_SIZE=4096,MAX_BLOB_PART_SIZE';
```

Setting the blob inline size in this fashion is supported by NDB 8.0.30 and later. Bear in mind that `ALTER TABLE ... COMMENT ...` discards any existing comment for the table. See Setting NDB_TABLE options, for additional information and examples.

- `ENGINE_ATTRIBUTE` and `SECONDARY_ENGINE_ATTRIBUTE` options (available as of MySQL 8.0.21) are used to specify table, column, and index attributes for primary and secondary storage engines. The options are reserved for future use. Index attributes cannot be altered. An index must be dropped and added back with the desired change, which can be performed in a single `ALTER TABLE` statement.

To verify that the table options were changed as intended, use `SHOW CREATE TABLE`, or query the Information Schema `TABLES` table.

Performance and Space Requirements

`ALTER TABLE` operations are processed using one of the following algorithms:

- `COPY`: Operations are performed on a copy of the original table, and table data is copied from the original table to the new table row by row. Concurrent DML is not permitted.
- `INPLACE`: Operations avoid copying table data but may rebuild the table in place. An exclusive metadata lock on the table may be taken briefly during preparation and execution phases of the operation. Typically, concurrent DML is supported.
- `INSTANT`: Operations only modify metadata in the data dictionary. An exclusive metadata lock on the table may be taken briefly during the execution phase of the operation. Table data is unaffected, making operations instantaneous. Concurrent DML is permitted. (Introduced in MySQL 8.0.12)

For tables using the `NDB` storage engine, these algorithms work as follows:

- `COPY`: `NDB` creates a copy of the table and alters it; the NDB Cluster handler then copies the data between the old and new versions of the table. Subsequently, `NDB` deletes the old table and renames the new one.

  This is sometimes also referred to as a “copying” or “offline” `ALTER TABLE`.

- `INPLACE`: The data nodes make the required changes; the NDB Cluster handler does not copy data or otherwise take part.

This is sometimes also referred to as a “non-copying” or “online” `ALTER TABLE`.

- `INSTANT`: Not supported by `NDB`.

See Section 23.6.12, “Online Operations with ALTER TABLE in NDB Cluster”, for more information.

The `ALGORITHM` clause is optional. If the `ALGORITHM` clause is omitted, MySQL uses `ALGORITHM=INSTANT` for storage engines and `ALTER TABLE` clauses that support it. Otherwise, `ALGORITHM=INPLACE` is used. If `ALGORITHM=INPLACE` is not supported, `ALGORITHM=COPY` is used.

> **Note**
>
> After adding a column to a partitioned table using `ALGORITHM=INSTANT`, it is no longer possible to perform `ALTER TABLE ... EXCHANGE PARTITION` on the table.

Specifying an `ALGORITHM` clause requires the operation to use the specified algorithm for clauses and storage engines that support it, or fail with an error otherwise. Specifying `ALGORITHM=DEFAULT` is the same as omitting the `ALGORITHM` clause.

`ALTER TABLE` operations that use the `COPY` algorithm wait for other operations that are modifying the table to complete. After alterations are applied to the table copy, data is copied over, the original table is deleted, and the table copy is renamed to the name of the original table. While the `ALTER TABLE` operation executes, the original table is readable by other sessions (with the exception noted shortly). Updates and writes to the table started after the `ALTER TABLE` operation begins are stalled until the new table is ready, then are automatically redirected to the new table. The temporary copy of the table is created in the database directory of the original table unless it is a `RENAME TO` operation that moves the table to a database that resides in a different directory.

The exception referred to earlier is that `ALTER TABLE` blocks reads (not just writes) at the point where it is ready to clear outdated table structures from the table and table definition caches. At this point, it must acquire an exclusive lock. To do so, it waits for current readers to finish, and blocks new reads and writes.

An `ALTER TABLE` operation that uses the `COPY` algorithm prevents concurrent DML operations. Concurrent queries are still allowed. That is, a table-copying operation always includes at least the concurrency restrictions of `LOCK=SHARED` (allow queries but not DML). You can further restrict concurrency for operations that support the `LOCK` clause by specifying `LOCK=EXCLUSIVE`, which prevents DML and queries. For more information, see Concurrency Control.

To force use of the `COPY` algorithm for an `ALTER TABLE` operation that would otherwise not use it, specify `ALGORITHM=COPY` or enable the `old_alter_table` system variable. If there is a conflict between the

`old_alter_table` setting and an `ALGORITHM` clause with a value other than `DEFAULT`, the `ALGORITHM` clause takes precedence.

For `InnoDB` tables, an `ALTER TABLE` operation that uses the `COPY` algorithm on a table that resides in a shared tablespace can increase the amount of space used by the tablespace. Such operations require as much additional space as the data in the table plus indexes. For a table residing in a shared tablespace, the additional space used during the operation is not released back to the operating system as it is for a table that resides in a file-per-table tablespace.

For information about space requirements for online DDL operations, see Section 15.12.3, "Online DDL Space Requirements".

`ALTER TABLE` operations that support the `INPLACE` algorithm include:

- `ALTER TABLE` operations supported by the `InnoDB` online DDL feature. See Section 15.12.1, "Online DDL Operations".
- Renaming a table. MySQL renames files that correspond to the table ***`tbl_name`*** without making a copy. (You can also use the `RENAME TABLE` statement to rename tables. See Section 13.1.36, "RENAME TABLE Statement".) Privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.
- Operations that modify table metadata only. These operations are immediate because the server does not touch table contents. Metadata-only operations include:
  - Renaming a column. In NDB Cluster 8.0.18 and later, this operation can also be performed online.
  - Changing the default value of a column (except for `NDB` tables).
  - Modifying the definition of an `ENUM` or `SET` column by adding new enumeration or set members to the *end* of the list of valid member values, as long as the storage size of the data type does not change. For example, adding a member to a `SET` column that has 8 members changes the required storage per value from 1 byte to 2 bytes; this requires a table copy. Adding members in the middle of the list causes renumbering of existing members, which requires a table copy.
  - Changing the definition of a spatial column to remove the `SRID` attribute. (Adding or changing an `SRID` attribute requires a rebuild, and cannot be done in place, because the server must verify that all values have the specified `SRID` value.)
  - As of MySQL 8.0.14, changing a column character set, when these conditions apply:

        - The column data type is `CHAR`, `VARCHAR`, a `TEXT` type, or `ENUM`.
        - The character set change is from `utf8mb3` to `utf8mb4`, or any character set to `binary`.
        - There is no index on the column.
    - As of MySQL 8.0.14, changing a generated column, when these conditions apply:
        - For `InnoDB` tables, statements that modify generated stored columns but do not change their type, expression, or nullability.
        - For non-`InnoDB` tables, statements that modify generated stored or virtual columns but do not change their type, expression, or nullability.

    An example of such a change is a change to the column comment.
- Renaming an index.
- Adding or dropping a secondary index, for `InnoDB` and `NDB` tables. See Section 15.12.1, “Online DDL Operations”.
- For `NDB` tables, operations that add and drop indexes on variable-width columns. These operations occur online, without table copying and without blocking concurrent DML actions for most of their duration. See Section 23.6.12, “Online Operations with ALTER TABLE in NDB Cluster”.
- Modifying index visibility with an `ALTER INDEX` operation.
- Column modifications of tables containing generated columns that depend on columns with a `DEFAULT` value if the modified columns are not involved in the generated column expressions. For example, changing the `NULL` property of a separate column can be done in place without a table rebuild.

`ALTER TABLE` operations that support the `INSTANT` algorithm include:

- Adding a column. This feature is referred to as “Instant `ADD COLUMN`”. Limitations apply. See Section 15.12.1, “Online DDL Operations”.
- Dropping a column. This feature is referred to as “Instant `DROP COLUMN`”. Limitations apply. See Section 15.12.1, “Online DDL Operations”.
- Adding or dropping a virtual column.
- Adding or dropping a column default value.
- Modifying the definition of an `ENUM` or `SET` column. The same restrictions apply as described above for `ALGORITHM=INSTANT`.

- Changing the index type.
- Renaming a table. The same restrictions apply as described above for `ALGORITHM=INSTANT`.

For more information about operations that support `ALGORITHM=INSTANT`, see Section 15.12.1, “Online DDL Operations”.

`ALTER TABLE` upgrades MySQL 5.5 temporal columns to 5.6 format for `ADD COLUMN`, `CHANGE COLUMN`, `MODIFY COLUMN`, `ADD INDEX`, and `FORCE` operations. This conversion cannot be done using the `INPLACE` algorithm because the table must be rebuilt, so specifying `ALGORITHM=INPLACE` in these cases results in an error. Specify `ALGORITHM=COPY` if necessary.

If an `ALTER TABLE` operation on a multicolumn index used to partition a table by `KEY` changes the order of the columns, it can only be performed using `ALGORITHM=COPY`.

The `WITHOUT VALIDATION` and `WITH VALIDATION` clauses affect whether `ALTER TABLE` performs an in-place operation for virtual generated column modifications. See Section 13.1.9.2, “ALTER TABLE and Generated Columns”.

NDB Cluster 8.0 supports online operations using the same `ALGORITHM=INPLACE` syntax used with the standard MySQL Server. `NDB` does not support changing a tablespace online; beginning with NDB 8.0.21, it is disallowed. See Section 23.6.12, “Online Operations with ALTER TABLE in NDB Cluster”, for more information.

NDB 8.0.27 and later, when performing a copying `ALTER TABLE`, checks to ensure that no concurrent writes have been made to the affected table. If it finds that any have been made, `NDB` rejects the `ALTER TABLE` statement and raises `ER_TABLE_DEF_CHANGED`.

`ALTER TABLE` with `DISCARD ... PARTITION ... TABLESPACE` or `IMPORT ... PARTITION ... TABLESPACE` does not create any temporary tables or temporary partition files.

`ALTER TABLE` with `ADD PARTITION`, `DROP PARTITION`, `COALESCE PARTITION`, `REBUILD PARTITION`, or `REORGANIZE PARTITION` does not create temporary tables (except when used with `NDB` tables); however, these operations can and do create temporary partition files.

`ADD` or `DROP` operations for `RANGE` or `LIST` partitions are immediate operations or nearly so. `ADD` or `COALESCE` operations for `HASH` or `KEY` partitions copy data between all partitions, unless `LINEAR HASH` or `LINEAR KEY` was used; this is effectively the same as creating a new table, although the `ADD` or `COALESCE` operation is performed partition by partition. `REORGANIZE` operations copy only changed partitions and do not touch unchanged ones.

For `MyISAM` tables, you can speed up index re-creation (the slowest part of the alteration process) by setting the `myisam_sort_buffer_size` system variable to a high value.

Concurrency Control

For `ALTER TABLE` operations that support it, you can use the `LOCK` clause to control the level of concurrent reads and writes on a table while it is being altered. Specifying a non-default value for this clause enables you to require a certain amount of concurrent access or exclusivity during the alter operation, and halts the operation if the requested degree of locking is not available.

Only `LOCK = DEFAULT` is permitted for operations that use `ALGORITHM=INSTANT`. The other `LOCK` clause parameters are not applicable.

The parameters for the `LOCK` clause are:

- `LOCK = DEFAULT`

  Maximum level of concurrency for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation: Permit concurrent reads and writes if supported. If not, permit concurrent reads if supported. If not, enforce exclusive access.

- `LOCK = NONE`

  If supported, permit concurrent reads and writes. Otherwise, an error occurs.

- `LOCK = SHARED`

  If supported, permit concurrent reads but block writes. Writes are blocked even if concurrent writes are supported by the storage engine for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation. If concurrent reads are not supported, an error occurs.

- `LOCK = EXCLUSIVE`

  Enforce exclusive access. This is done even if concurrent reads/writes are supported by the storage engine for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation.

Adding and Dropping Columns

Use `ADD` to add new columns to a table, and `DROP` to remove existing columns. `DROP` ***`col_name`*** is a MySQL extension to standard SQL.

To add a column at a specific position within a table row, use `FIRST` or `AFTER col_name`. The default is to add the column last.

If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use the `DROP TABLE` statement instead.

If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well. If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.

For `ALTER TABLE ... ADD`, if the column has an expression default value that uses a nondeterministic function, the statement may produce a warning or error. For further information, see Section 11.6, “Data Type Default Values”, and Section 17.1.3.7, “Restrictions on Replication with GTIDs”.

Renaming, Redefining, and Reordering Columns

The `CHANGE`, `MODIFY`, `RENAME COLUMN`, and `ALTER` clauses enable the names and definitions of existing columns to be altered. They have these comparative characteristics:

- `CHANGE`:
  - Can rename a column and change its definition, or both.
  - Has more capability than `MODIFY` or `RENAME COLUMN`, but at the expense of convenience for some operations. `CHANGE` requires naming the column twice if not renaming it, and requires respecifying the column definition if only renaming it.
  - With `FIRST` or `AFTER`, can reorder columns.
- `MODIFY`:
  - Can change a column definition but not its name.
  - More convenient than `CHANGE` to change a column definition without renaming it.
  - With `FIRST` or `AFTER`, can reorder columns.
- `RENAME COLUMN`:
  - Can change a column name but not its definition.
  - More convenient than `CHANGE` to rename a column without changing its definition.

- `ALTER`: Used only to change a column default value.

`CHANGE` is a MySQL extension to standard SQL. `MODIFY` and `RENAME COLUMN` are MySQL extensions for Oracle compatibility.

To alter a column to change both its name and definition, use `CHANGE`, specifying the old and new names and the new definition. For example, to rename an `INT NOT NULL` column from `a` to `b` and change its definition to use the `BIGINT` data type while retaining the `NOT NULL` attribute, do this:

```
ALTER TABLE t1 CHANGE a b BIGINT NOT NULL;
```

To change a column definition but not its name, use `CHANGE` or `MODIFY`. With `CHANGE`, the syntax requires two column names, so you must specify the same name twice to leave the name unchanged. For example, to change the definition of column `b`, do this:

```
ALTER TABLE t1 CHANGE b b INT NOT NULL;
```

`MODIFY` is more convenient to change the definition without changing the name because it requires the column name only once:

```
ALTER TABLE t1 MODIFY b INT NOT NULL;
```

To change a column name but not its definition, use `CHANGE` or `RENAME COLUMN`. With `CHANGE`, the syntax requires a column definition, so to leave the definition unchanged, you must respecify the definition the column currently has. For example, to rename an `INT NOT NULL` column from `b` to `a`, do this:

```
ALTER TABLE t1 CHANGE b a INT NOT NULL;
```

`RENAME COLUMN` is more convenient to change the name without changing the definition because it requires only the old and new names:

```
ALTER TABLE t1 RENAME COLUMN b TO a;
```

In general, you cannot rename a column to a name that already exists in the table. However, this is sometimes not the case, such as when you swap names or move them through a cycle. If a table has

columns named `a`, `b`, and `c`, these are valid operations:

```
-- swap a and b
ALTER TABLE t1 RENAME COLUMN a TO b,
               RENAME COLUMN b TO a;
-- "rotate" a, b, c through a cycle
ALTER TABLE t1 RENAME COLUMN a TO b,
               RENAME COLUMN b TO c,
               RENAME COLUMN c TO a;
```

For column definition changes using `CHANGE` or `MODIFY`, the definition must include the data type and all attributes that should apply to the new column, other than index attributes such as `PRIMARY KEY` or `UNIQUE`. Attributes present in the original definition but not specified for the new definition are not carried forward. Suppose that a column `col1` is defined as `INT UNSIGNED DEFAULT 1 COMMENT 'my column'` and you modify the column as follows, intending to change only `INT` to `BIGINT`:

```
ALTER TABLE t1 MODIFY col1 BIGINT;
```

That statement changes the data type from `INT` to `BIGINT`, but it also drops the `UNSIGNED`, `DEFAULT`, and `COMMENT` attributes. To retain them, the statement must include them explicitly:

```
ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my column';
```

For data type changes using `CHANGE` or `MODIFY`, MySQL tries to convert existing column values to the new type as well as possible.

> **Warning**
>
> This conversion may result in alteration of data. For example, if you shorten a string column, values may be truncated. To prevent the operation from succeeding if conversions to the new data type would result in loss of data, enable strict SQL mode before using `ALTER TABLE` (see Section 5.1.11, “Server SQL Modes”).

If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.

For columns renamed by `CHANGE` or `RENAME COLUMN`, MySQL automatically renames these references to the renamed column:

- Indexes that refer to the old column, including invisible indexes and disabled `MyISAM` indexes.
- Foreign keys that refer to the old column.

For columns renamed by `CHANGE` or `RENAME COLUMN`, MySQL does not automatically rename these references to the renamed column:

- Generated column and partition expressions that refer to the renamed column. You must use `CHANGE` to redefine such expressions in the same `ALTER TABLE` statement as the one that renames the column.
- Views and stored programs that refer to the renamed column. You must manually alter the definition of these objects to refer to the new column name.

To reorder columns within a table, use `FIRST` and `AFTER` in `CHANGE` or `MODIFY` operations.

`ALTER ... SET DEFAULT` or `ALTER ... DROP DEFAULT` specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be `NULL`, the new default is `NULL`. If the column cannot be `NULL`, MySQL assigns a default value as described in Section 11.6, “Data Type Default Values”.

As of MySQL 8.0.23, `ALTER ... SET VISIBLE` and `ALTER ... SET INVISIBLE` enable column visibility to be changed. See Section 13.1.20.10, “Invisible Columns”.

Primary Keys and Indexes

`DROP PRIMARY KEY` drops the primary key. If there is no primary key, an error occurs. For information about the performance characteristics of primary keys, especially for `InnoDB` tables, see Section 8.3.2, “Primary Key Optimization”.

If the `sql_require_primary_key` system variable is enabled, attempting to drop a primary key produces an error.

If you add a `UNIQUE INDEX` or `PRIMARY KEY` to a table, MySQL stores it before any nonunique index to permit detection of duplicate keys as early as possible.

`DROP INDEX` removes an index. This is a MySQL extension to standard SQL. See Section 13.1.27, “DROP INDEX Statement”. To determine index names, use `SHOW INDEX FROM` ***`tbl_name`***.

Some storage engines permit you to specify an index type when creating an index. The syntax for the ***index_type*** specifier is `USING type_name`. For details about `USING`, see Section 13.1.15, “CREATE INDEX Statement”. The preferred position is after the column list. Expect support for use of the option before the column list to be removed in a future MySQL release.

index_option values specify additional options for an index. `USING` is one such option. For details about permissible ***index_option*** values, see Section 13.1.15, “CREATE INDEX Statement”.

`RENAME INDEX old_index_name TO new_index_name` renames an index. This is a MySQL extension to standard SQL. The content of the table remains unchanged. ***old_index_name*** must be the name of an existing index in the table that is not dropped by the same `ALTER TABLE` statement. ***new_index_name*** is the new index name, which cannot duplicate the name of an index in the resulting table after changes have been applied. Neither index name can be `PRIMARY`.

If you use `ALTER TABLE` on a `MyISAM` table, all nonunique indexes are created in a separate batch (as for `REPAIR TABLE`). This should make `ALTER TABLE` much faster when you have many indexes.

For `MyISAM` tables, key updating can be controlled explicitly. Use `ALTER TABLE ... DISABLE KEYS` to tell MySQL to stop updating nonunique indexes. Then use `ALTER TABLE ... ENABLE KEYS` to re-create missing indexes. `MyISAM` does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using `ALTER TABLE ... DISABLE KEYS` requires the `INDEX` privilege in addition to the privileges mentioned earlier.

While the nonunique indexes are disabled, they are ignored for statements such as `SELECT` and `EXPLAIN` that otherwise would use them.

After an `ALTER TABLE` statement, it may be necessary to run `ANALYZE TABLE` to update index cardinality information. See Section 13.7.7.22, “SHOW INDEX Statement”.

The `ALTER INDEX` operation permits an index to be made visible or invisible. An invisible index is not used by the optimizer. Modification of index visibility applies to indexes other than primary keys (either explicit or implicit). This feature is storage engine neutral (supported for any engine). For more information, see Section 8.3.12, “Invisible Indexes”.

Foreign Keys and Other Constraints

The `FOREIGN KEY` and `REFERENCES` clauses are supported by the `InnoDB` and `NDB` storage engines, which implement `ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (...) REFERENCES ... (...)`.

See Section 13.1.20.5, “FOREIGN KEY Constraints”. For other storage engines, the clauses are parsed but ignored.

For `ALTER TABLE`, unlike `CREATE TABLE`, `ADD FOREIGN KEY` ignores ***`index_name`*** if given and uses an automatically generated foreign key name. As a workaround, include the `CONSTRAINT` clause to specify the foreign key name:

```
ADD CONSTRAINT name FOREIGN KEY (....) ...
```

> **Important**
>
> MySQL silently ignores inline `REFERENCES` specifications, where the references are defined as part of the column specification. MySQL accepts only `REFERENCES` clauses defined as part of a separate `FOREIGN KEY` specification.

> **Note**
>
> Partitioned `InnoDB` tables do not support foreign keys. This restriction does not apply to `NDB` tables, including those explicitly partitioned by `[LINEAR] KEY`. For more information, see Section 24.6.2, “Partitioning Limitations Relating to Storage Engines”.

MySQL Server and NDB Cluster both support the use of `ALTER TABLE` to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

Adding and dropping a foreign key in the same `ALTER TABLE` statement is supported for `ALTER TABLE ... ALGORITHM=INPLACE` but not for `ALTER TABLE ... ALGORITHM=COPY`.

The server prohibits changes to foreign key columns that have the potential to cause loss of referential integrity. A workaround is to use `ALTER TABLE ... DROP FOREIGN KEY` before changing the column definition and `ALTER TABLE ... ADD FOREIGN KEY` afterward. Examples of prohibited changes include:

- Changes to the data type of foreign key columns that may be unsafe. For example, changing `VARCHAR(20)` to `VARCHAR(30)` is permitted, but changing it to `VARCHAR(1024)` is not because that alters the number of length bytes required to store individual values.

- Changing a NULL column to NOT NULL in non-strict mode is prohibited to prevent converting NULL values to default non-NULL values, for which there are no corresponding values in the referenced table. The operation is permitted in strict mode, but an error is returned if any such conversion is required.

ALTER TABLE ***tbl_name*** RENAME ***new_tbl_name*** changes internally generated foreign key constraint names and user-defined foreign key constraint names that begin with the string "***tbl_name***_ibfk_" to reflect the new table name. InnoDB interprets foreign key constraint names that begin with the string "***tbl_name***_ibfk_" as internally generated names.

Prior to MySQL 8.0.16, ALTER TABLE permits only the following limited version of CHECK constraint-adding syntax, which is parsed and ignored:

```
ADD CHECK (expr)
```

As of MySQL 8.0.16, ALTER TABLE permits CHECK constraints for existing tables to be added, dropped, or altered:

- Add a new CHECK constraint:

  ```
  ALTER TABLE tbl_name
      ADD [CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED];
  ```

  The meaning of constraint syntax elements is the same as for CREATE TABLE. See Section 13.1.20.6, "CHECK Constraints".

- Drop an existing CHECK constraint named ***symbol***:

  ```
  ALTER TABLE tbl_name
      DROP CHECK symbol;
  ```

- Alter whether an existing CHECK constraint named ***symbol*** is enforced:

  ```
  ALTER TABLE tbl_name
      ALTER CHECK symbol [NOT] ENFORCED;
  ```

The DROP CHECK and ALTER CHECK clauses are MySQL extensions to standard SQL.

As of MySQL 8.0.19, `ALTER TABLE` permits more general (and SQL standard) syntax for dropping and altering existing constraints of any type, where the constraint type is determined from the constraint name:

- Drop an existing constraint named ***`symbol`***:

  ```
  ALTER TABLE tbl_name
      DROP CONSTRAINT symbol;
  ```

  If the `sql_require_primary_key` system variable is enabled, attempting to drop a primary key produces an error.

- Alter whether an existing constraint named ***`symbol`*** is enforced:

  ```
  ALTER TABLE tbl_name
      ALTER CONSTRAINT symbol [NOT] ENFORCED;
  ```

  Only `CHECK` constraints can be altered to be unenforced. All other constraint types are always enforced.

The SQL standard specifies that all types of constraints (primary key, unique index, foreign key, check) belong to the same namespace. In MySQL, each constraint type has its own namespace per schema. Consequently, names for each type of constraint must be unique per schema, but constraints of different types can have the same name. When multiple constraints have the same name, `DROP CONSTRAINT` and `ADD CONSTRAINT` are ambiguous and an error occurs. In such cases, constraint-specific syntax must be used to modify the constraint. For example, use `DROP PRIMARY KEY` or DROP FOREIGN KEY to drop a primary key or foreign key.

If a table alteration causes a violation of an enforced `CHECK` constraint, an error occurs and the table is not modified. Examples of operations for which an error occurs:

- Attempts to add the `AUTO_INCREMENT` attribute to a column that is used in a `CHECK` constraint.
- Attempts to add an enforced `CHECK` constraint or enforce a nonenforced `CHECK` constraint for which existing rows violate the constraint condition.
- Attempts to modify, rename, or drop a column that is used in a `CHECK` constraint, unless that constraint is also dropped in the same statement. Exception: If a `CHECK` constraint refers only to a single column, dropping the column automatically drops the constraint.

`ALTER TABLE` ***`tbl_name`*** `RENAME` ***`new_tbl_name`*** changes internally generated and user-defined `CHECK` constraint names that begin with the string "***`tbl_name`***_chk_" to reflect the new table name. MySQL interprets `CHECK` constraint names that begin with the string "***`tbl_name`***_chk_" as internally generated names.

Changing the Character Set

To change the table default character set and all character columns (`CHAR`, `VARCHAR`, `TEXT`) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

The statement also changes the collation of all character columns. If you specify no `COLLATE` clause to indicate which collation to use, the statement uses default collation for the character set. If this collation is inappropriate for the intended table use (for example, if it would change from a case-sensitive collation to a case-insensitive collation), specify a collation explicitly.

For a column that has a data type of `VARCHAR` or one of the `TEXT` types, `CONVERT TO CHARACTER SET` changes the data type as necessary to ensure that the new column is long enough to store as many characters as the original column. For example, a `TEXT` column has two length bytes, which store the byte-length of values in the column, up to a maximum of 65,535. For a `latin1` `TEXT` column, each character requires a single byte, so the column can store up to 65,535 characters. If the column is converted to `utf8mb4`, each character might require up to 4 bytes, for a maximum possible length of 4 × 65,535 = 262,140 bytes. That length does not fit in a `TEXT` column's length bytes, so MySQL converts the data type to `MEDIUMTEXT`, which is the smallest string type for which the length bytes can record a value of 262,140. Similarly, a `VARCHAR` column might be converted to `MEDIUMTEXT`.

To avoid data type changes of the type just described, do not use `CONVERT TO CHARACTER SET`. Instead, use `MODIFY` to change individual columns. For example:

```
ALTER TABLE t MODIFY latin1_text_col TEXT CHARACTER SET utf8mb4;
ALTER TABLE t MODIFY latin1_varchar_col VARCHAR(M) CHARACTER SET utf8mb4;
```

If you specify `CONVERT TO CHARACTER SET binary`, the `CHAR`, `VARCHAR`, and `TEXT` columns are converted to their corresponding binary string types (`BINARY`, `VARBINARY`, `BLOB`). This means that the columns no longer have a character set and a subsequent `CONVERT TO` operation does not apply to them.

If ***charset_name*** is `DEFAULT` in a `CONVERT TO CHARACTER SET` operation, the character set named by the `character_set_database` system variable is used.

> **Warning**
>
> The `CONVERT TO` operation converts column values between the original and named character sets. This is *not* what you want if you have a column in one character set (like `latin1`) but the stored values actually use some other, incompatible character set (like `utf8mb4`). In this case, you have to do the following for each such column:
>
> ```
> ALTER TABLE t1 CHANGE c1 c1 BLOB;
> ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8mb4;
> ```
>
> The reason this works is that there is no conversion when you convert to or from `BLOB` columns.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word `DEFAULT` is optional. The default character set is the character set that is used if you do not specify the character set for columns that you add to a table later (for example, with `ALTER TABLE ... ADD column`).

When the `foreign_key_checks` system variable is enabled, which is the default setting, character set conversion is not permitted on tables that include a character string column used in a foreign key constraint. The workaround is to disable `foreign_key_checks` before performing the character set conversion. You must perform the conversion on both tables involved in the foreign key constraint before re-enabling `foreign_key_checks`. If you re-enable `foreign_key_checks` after converting only one of the tables, an `ON DELETE CASCADE` or `ON UPDATE CASCADE` operation could corrupt data in the referencing table due to implicit conversion that occurs during these operations (Bug #45290, Bug #74816).

Importing InnoDB Tables

An `InnoDB` table created in its own file-per-table tablespace can be imported from a backup or from another MySQL server instance using `DISCARD TABLEPACE` and `IMPORT TABLESPACE` clauses. See

Section 15.6.1.3, “Importing InnoDB Tables”.

Row Order for MyISAM Tables

`ORDER BY` enables you to create the new table with the rows in a specific order. This option is useful primarily when you know that you query the rows in a certain order most of the time. By using this option after major changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.

> **Note**
>
> The table does not remain in the specified order after inserts and deletes.

`ORDER BY` syntax permits one or more column names to be specified for sorting, each of which optionally can be followed by `ASC` or `DESC` to indicate ascending or descending sort order, respectively. The default is ascending order. Only column names are permitted as sort criteria; arbitrary expressions are not permitted. This clause should be given last after any other clauses.

`ORDER BY` does not make sense for `InnoDB` tables because `InnoDB` always orders table rows according to the clustered index.

When used on a partitioned table, `ALTER TABLE ... ORDER BY` orders rows within each partition only.

Partitioning Options

`partition_options` signifies options that can be used with partitioned tables for repartitioning, to add, drop, discard, import, merge, and split partitions, and to perform partitioning maintenance.

It is possible for an `ALTER TABLE` statement to contain a `PARTITION BY` or `REMOVE PARTITIONING` clause in an addition to other alter specifications, but the `PARTITION BY` or `REMOVE PARTITIONING` clause must be specified last after any other specifications. The `ADD PARTITION`, `DROP PARTITION`, `DISCARD PARTITION`, `IMPORT PARTITION`, `COALESCE PARTITION`, `REORGANIZE PARTITION`, `EXCHANGE PARTITION`, `ANALYZE PARTITION`, `CHECK PARTITION`, and `REPAIR PARTITION` options cannot be combined with other alter specifications in a single `ALTER TABLE`, since the options just listed act on individual partitions.

For more information about partition options, see Section 13.1.20, “CREATE TABLE Statement”, and Section 13.1.9.1, “ALTER TABLE Partition Operations”. For information about and examples of `ALTER TABLE ... EXCHANGE PARTITION` statements, see Section 24.3.3, “Exchanging Partitions and Subpartitions with Tables”.