14.13.1 Online DDL Operations

Online support details, syntax examples, and usage notes for DDL operations are provided under the following topics in this section.

- Index Operations
- Primary Key Operations
- Column Operations
- Generated Column Operations
- Foreign Key Operations
- Table Operations
- Tablespace Operations
- Partitioning Operations

Index Operations

The following table provides an overview of online DDL support for index operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 14.10 Online DDL Support for Index Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Creating or adding a secondary index	Yes	No	Yes	No
Dropping an index	Yes	No	Yes	Yes
Renaming an index	Yes	No	Yes	Yes
Adding a `FULLTEXT` index	Yes*	No*	No	No
Adding a `SPATIAL` index	Yes	No	No	No
Changing the index type	Yes	No	Yes	Yes

Syntax and Usage Notes

- Creating or adding a secondary index

```
CREATE INDEX name ON table (col_list);
```

```
ALTER TABLE tbl_name ADD INDEX name (col_list);
```

The table remains available for read and write operations while the index is being created. The `CREATE INDEX` statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table.

Online DDL support for adding secondary indexes means that you can generally speed the overall process of creating and loading a table and associated indexes by creating the table without secondary indexes, then adding secondary indexes after the data is loaded.

A newly created secondary index contains only the committed data in the table at the time the `CREATE INDEX` or `ALTER TABLE` statement finishes executing. It does not contain any uncommitted values, old versions of values, or values marked for deletion but not yet removed from the old index.

If the server exits while creating a secondary index, upon recovery, MySQL drops any partially created indexes. You must re-run the `ALTER TABLE` or `CREATE INDEX` statement.

Some factors affect the performance, space usage, and semantics of this operation. For details, see Section 14.13.6, “Online DDL Limitations”.

- Dropping an index

```
DROP INDEX name ON table;
```

```
ALTER TABLE tbl_name DROP INDEX name;
```

The table remains available for read and write operations while the index is being dropped. The `DROP INDEX` statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table.

- Renaming an index

```
ALTER TABLE tbl_name RENAME INDEX old_index_name TO new_index_name, ALGORITHM=INPLACE, LOCK=NONE;
```

- Adding a `FULLTEXT` index

```
CREATE FULLTEXT INDEX name ON table(column);
```

Adding the first FULLTEXT index rebuilds the table if there is no user-defined FTS_DOC_ID column. Additional FULLTEXT indexes may be added without rebuilding the table.

- Adding a SPATIAL index

```
CREATE TABLE geom (g GEOMETRY NOT NULL);
ALTER TABLE geom ADD SPATIAL INDEX(g), ALGORITHM=INPLACE, LOCK=SHARED;
```

- Changing the index type (USING {BTREE | HASH})

```
ALTER TABLE tbl_name DROP INDEX i1, ADD INDEX i1(key_part,...) USING BTREE, ALGORITHM=INPLACE;
```

Primary Key Operations

The following table provides an overview of online DDL support for primary key operations. An asterisk indicates additional information, an exception, or a dependency. See Syntax and Usage Notes.

Table 14.11 Online DDL Support for Primary Key Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a primary key	Yes*	Yes*	Yes	No
Dropping a primary key	No	Yes	No	No
Dropping a primary key and adding another	Yes	Yes	Yes	No

Syntax and Usage Notes

- Adding a primary key

```
ALTER TABLE tbl_name ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

Rebuilds the table in place. Data is reorganized substantially, making it an expensive operation. `ALGORITHM=INPLACE` is not permitted under certain conditions if columns have to be converted to `NOT NULL`.

Restructuring the clustered index always requires copying of table data. Thus, it is best to define the primary key when you create a table, rather than issuing `ALTER TABLE ... ADD PRIMARY KEY` later.

When you create a `UNIQUE` or `PRIMARY KEY` index, MySQL must do some extra work. For `UNIQUE` indexes, MySQL checks that the table contains no duplicate values for the key. For a `PRIMARY KEY` index, MySQL also checks that none of the `PRIMARY KEY` columns contains a `NULL`.

When you add a primary key using the `ALGORITHM=COPY` clause, MySQL converts `NULL` values in the associated columns to default values: 0 for numbers, an empty string for character-based columns and BLOBs, and 0000-00-00 00:00:00 for `DATETIME`. This is a non-standard behavior that Oracle recommends you not rely on. Adding a primary key using `ALGORITHM=INPLACE` is only permitted when the `SQL_MODE` setting includes the `strict_trans_tables` or `strict_all_tables` flags; when the `SQL_MODE` setting is strict, `ALGORITHM=INPLACE` is permitted, but the statement can still fail if the requested primary key columns contain `NULL` values. The `ALGORITHM=INPLACE` behavior is more standard-compliant.

If you create a table without a primary key, `InnoDB` chooses one for you, which can be the first `UNIQUE` key defined on `NOT NULL` columns, or a system-generated key. To avoid uncertainty and the potential space requirement for an extra hidden column, specify the `PRIMARY KEY` clause as part of the `CREATE TABLE` statement.

MySQL creates a new clustered index by copying the existing data from the original table to a temporary table that has the desired index structure. Once the data is completely copied to the temporary table, the original table is renamed with a different temporary table name. The temporary table comprising the new clustered index is renamed with the name of the original table, and the original table is dropped from the database.

The online performance enhancements that apply to operations on secondary indexes do not apply to the primary key index. The rows of an InnoDB table are stored in a clustered index organized based on the primary key, forming what some database systems call an “index-organized table”. Because the table structure is closely tied to the primary key, redefining the primary key still requires copying the data.

When an operation on the primary key uses `ALGORITHM=INPLACE`, even though the data is still copied, it is more efficient than using `ALGORITHM=COPY` because:

- No undo logging or associated redo logging is required for `ALGORITHM=INPLACE`. These operations add overhead to DDL statements that use `ALGORITHM=COPY`.
- The secondary index entries are pre-sorted, and so can be loaded in order.
- The change buffer is not used, because there are no random-access inserts into the secondary indexes.

If the server exits while creating a new clustered index, no data is lost, but you must complete the recovery process using the temporary tables that exist during the process. Since it is rare to re-create a clustered index or re-define primary keys on large tables, or to encounter a system crash during this operation, this manual does not provide information on recovering from this scenario.

- Dropping a primary key

```
ALTER TABLE tbl_name DROP PRIMARY KEY, ALGORITHM=COPY;
```

Only `ALGORITHM=COPY` supports dropping a primary key without adding a new one in the same `ALTER TABLE` statement.

- Dropping a primary key and adding another

```
ALTER TABLE tbl_name DROP PRIMARY KEY, ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

Data is reorganized substantially, making it an expensive operation.

Column Operations

The following table provides an overview of online DDL support for column operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 14.12 Online DDL Support for Column Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a column	Yes	Yes	Yes*	No
Dropping a column	Yes	Yes	Yes	No
Renaming a column	Yes	No	Yes*	Yes
Reordering columns	Yes	Yes	Yes	No
Setting a column default value	Yes	No	Yes	Yes
Changing the column data type	No	Yes	No	No
Extending VARCHAR **column size**	Yes	No	Yes	Yes
Dropping the column default value	Yes	No	Yes	Yes
Changing the auto-increment value	Yes	No	Yes	No*
Making a column `NULL`	Yes	Yes*	Yes	No
Making a column `NOT NULL`	Yes*	Yes*	Yes	No

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Modifying the definition of an `ENUM` or `SET` column	Yes	No	Yes	Yes

Syntax and Usage Notes

- Adding a column

```
ALTER TABLE tbl_name ADD COLUMN column_name column_definition, ALGORITHM=INPLACE, LOCK=NONE;
```

  Concurrent DML is not permitted when adding an auto-increment column. Data is reorganized substantially, making it an expensive operation. At a minimum, `ALGORITHM=INPLACE, LOCK=SHARED` is required.

- Dropping a column

```
ALTER TABLE tbl_name DROP COLUMN column_name, ALGORITHM=INPLACE, LOCK=NONE;
```

  Data is reorganized substantially, making it an expensive operation.

- Renaming a column

```
ALTER TABLE tbl CHANGE old_col_name new_col_name data_type, ALGORITHM=INPLACE, LOCK=NONE;
```

  To permit concurrent DML, keep the same data type and only change the column name.

When you keep the same data type and `[NOT] NULL` attribute, only changing the column name, the operation can always be performed online.

You can also rename a column that is part of a foreign key constraint. The foreign key definition is automatically updated to use the new column name. Renaming a column participating in a foreign key only works with `ALGORITHM=INPLACE`. If you use the `ALGORITHM=COPY` clause, or some other condition causes the operation to use `ALGORITHM=COPY`, the `ALTER TABLE` statement fails.

`ALGORITHM=INPLACE` is not supported for renaming a generated column.

- Reordering columns

  To reorder columns, use `FIRST` or `AFTER` in `CHANGE` or `MODIFY` operations.

  ```
  ALTER TABLE tbl_name MODIFY COLUMN col_name column_definition FIRST, ALGORITHM=INPLACE, LOCK=NONE;
  ```

  Data is reorganized substantially, making it an expensive operation.

- Changing the column data type

  ```
  ALTER TABLE tbl_name CHANGE c1 c1 BIGINT, ALGORITHM=COPY;
  ```

  Changing the column data type is only supported with `ALGORITHM=COPY`.

- Extending `VARCHAR` column size

  ```
  ALTER TABLE tbl_name CHANGE COLUMN c1 c1 VARCHAR(255), ALGORITHM=INPLACE, LOCK=NONE;
  ```

The number of length bytes required by a VARCHAR column must remain the same. For VARCHAR columns of 0 to 255 bytes in size, one length byte is required to encode the value. For VARCHAR columns of 256 bytes in size or more, two length bytes are required. As a result, in-place ALTER TABLE only supports increasing VARCHAR column size from 0 to 255 bytes, or from 256 bytes to a greater size. In-place ALTER TABLE does not support increasing the size of a VARCHAR column from less than 256 bytes to a size equal to or greater than 256 bytes. In this case, the number of required length bytes changes from 1 to 2, which is only supported by a table copy (ALGORITHM=COPY). For example, attempting to change VARCHAR column size for a single byte character set from VARCHAR(255) to VARCHAR(256) using in-place ALTER TABLE returns this error:

```
ALTER TABLE tbl_name ALGORITHM=INPLACE, CHANGE COLUMN c1 c1 VARCHAR(256);
ERROR 0A000: ALGORITHM=INPLACE is not supported. Reason: Cannot change
column type INPLACE. Try ALGORITHM=COPY.
```

> **Note**
>
> The byte length of a VARCHAR column is dependant on the byte length of the character set.

Decreasing VARCHAR size using in-place ALTER TABLE is not supported. Decreasing VARCHAR size requires a table copy (ALGORITHM=COPY).

- Setting a column default value

```
ALTER TABLE tbl_name ALTER COLUMN col SET DEFAULT literal, ALGORITHM=INPLACE, LOCK=NONE;
```

Only modifies table metadata. Default column values are stored in the .frm file for the table, not the InnoDB data dictionary.

- Dropping a column default value

```
ALTER TABLE tbl ALTER COLUMN col DROP DEFAULT, ALGORITHM=INPLACE, LOCK=NONE;
```

- Changing the auto-increment value

```
ALTER TABLE table AUTO_INCREMENT=next_value, ALGORITHM=INPLACE, LOCK=NONE;
```

  Modifies a value stored in memory, not the data file.

  In a distributed system using replication or sharding, you sometimes reset the auto-increment counter for a table to a specific value. The next row inserted into the table uses the specified value for its auto-increment column. You might also use this technique in a data warehousing environment where you periodically empty all the tables and reload them, and restart the auto-increment sequence from 1.

- Making a column `NULL`

```
ALTER TABLE tbl_name MODIFY COLUMN column_name data_type NULL, ALGORITHM=INPLACE, LOCK=NONE;
```

  Rebuilds the table in place. Data is reorganized substantially, making it an expensive operation.

- Making a column `NOT NULL`

```
ALTER TABLE tbl_name MODIFY COLUMN column_name data_type NOT NULL, ALGORITHM=INPLACE, LOCK=NONE;
```

  Rebuilds the table in place. `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` `SQL_MODE` is required for the operation to succeed. The operation fails if the column contains NULL values. The server prohibits changes to foreign key columns that have the potential to cause loss of referential integrity. See Section 13.1.8, “ALTER TABLE Statement”. Data is reorganized substantially, making it an expensive operation.

- Modifying the definition of an `ENUM` or `SET` column

```
CREATE TABLE t1 (c1 ENUM('a', 'b', 'c'));
ALTER TABLE t1 MODIFY COLUMN c1 ENUM('a', 'b', 'c', 'd'), ALGORITHM=INPLACE, LOCK=NONE;
```

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 may be performed in place, 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.

Generated Column Operations

The following table provides an overview of online DDL support for generated column operations. For details, see Syntax and Usage Notes.

Table 14.13 Online DDL Support for Generated Column Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a `STORED` column	No	Yes	No	No
Modifying `STORED` column order	No	Yes	No	No
Dropping a `STORED` column	Yes	Yes	Yes	No
Adding a `VIRTUAL` column	Yes	No	Yes	Yes
Modifying `VIRTUAL` column order	No	Yes	No	No
Dropping a `VIRTUAL` column	Yes	No	Yes	Yes

Syntax and Usage Notes

- Adding a `STORED` column

```
ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) STORED), ALGORITHM=COPY;
```

`ADD COLUMN` is not an in-place operation for stored columns (done without using a temporary table) because the expression must be evaluated by the server.

- Modifying `STORED` column order

```
ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED FIRST, ALGORITHM=COPY;
```

Rebuilds the table in place.

- Dropping a `STORED` column

```
ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INPLACE, LOCK=NONE;
```

Rebuilds the table in place.

- Adding a `VIRTUAL` column

```
ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL), ALGORITHM=INPLACE, LOCK=NONE;
```

Adding a virtual column is an in-place operation for non-partitioned tables. However, adding a virtual column cannot be combined with other ALTER TABLE actions.

Adding a VIRTUAL is not an in-place operation for partitioned tables.

- Modifying VIRTUAL column order

```
ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL FIRST, ALGORITHM=COPY;
```

- Dropping a VIRTUAL column

```
ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INPLACE, LOCK=NONE;
```

Dropping a VIRTUAL column is an in-place operation for non-partitioned tables. However, dropping a virtual column cannot be combined with other ALTER TABLE actions.

Dropping a VIRTUAL is not an in-place operation for partitioned tables.

Foreign Key Operations

The following table provides an overview of online DDL support for foreign key operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 14.14 Online DDL Support for Foreign Key Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a foreign key constraint	Yes*	No	Yes	Yes
Dropping a foreign key constraint	Yes	No	Yes	Yes

Syntax and Usage Notes

- Adding a foreign key constraint

  The `INPLACE` algorithm is supported when `foreign_key_checks` is disabled. Otherwise, only the `COPY` algorithm is supported.

```
ALTER TABLE tbl1 ADD CONSTRAINT fk_name FOREIGN KEY index (col1)
  REFERENCES tbl2(col2) referential_actions;
```

- Dropping a foreign key constraint

```
ALTER TABLE tbl DROP FOREIGN KEY fk_name;
```

  Dropping a foreign key can be performed online with the `foreign_key_checks` option enabled or disabled.

  If you do not know the names of the foreign key constraints on a particular table, issue the following statement and find the constraint name in the `CONSTRAINT` clause for each foreign key:

```
SHOW CREATE TABLE table\G
```

Or, query the `INFORMATION_SCHEMA.TABLE_CONSTRAINTS` table and use the `CONSTRAINT_NAME` and `CONSTRAINT_TYPE` columns to identify the foreign key names.

You can also drop a foreign key and its associated index in a single statement:

```
ALTER TABLE table DROP FOREIGN KEY constraint, DROP INDEX index;
```

> **Note**
>
> If foreign keys are already present in the table being altered (that is, it is a child table containing a `FOREIGN KEY ... REFERENCE` clause), additional restrictions apply to online DDL operations, even those not directly involving the foreign key columns:
>
> - An `ALTER TABLE` on the child table could wait for another transaction to commit, if a change to the parent table causes associated changes in the child table through an `ON UPDATE` or `ON DELETE` clause using the `CASCADE` or `SET NULL` parameters.
> - In the same way, if a table is the parent table in a foreign key relationship, even though it does not contain any `FOREIGN KEY` clauses, it could wait for the `ALTER TABLE` to complete if an `INSERT`, `UPDATE`, or `DELETE` statement causes an `ON UPDATE` or `ON DELETE` action in the child table.

Table Operations

The following table provides an overview of online DDL support for table operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 14.15 Online DDL Support for Table Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Changing the `ROW_FORMAT`	Yes	Yes	Yes	No
Changing the `KEY_BLOCK_SIZE`	Yes	Yes	Yes	No
Setting persistent table statistics	Yes	No	Yes	Yes
Specifying a character set	Yes	Yes*	No	No
Converting a character set	No	Yes*	No	No
Optimizing a table	Yes*	Yes	Yes	No
Rebuilding with the `FORCE` **option**	Yes*	Yes	Yes	No
Performing a null rebuild	Yes*	Yes	Yes	No
Renaming a table	Yes	No	Yes	Yes

Syntax and Usage Notes

- Changing the `ROW_FORMAT`

```
ALTER TABLE tbl_name ROW_FORMAT = row_format, ALGORITHM=INPLACE, LOCK=NONE;
```

  Data is reorganized substantially, making it an expensive operation.

  For additional information about the `ROW_FORMAT` option, see Table Options.

- Changing the KEY_BLOCK_SIZE

```
ALTER TABLE tbl_name KEY_BLOCK_SIZE = value, ALGORITHM=INPLACE, LOCK=NONE;
```

  Data is reorganized substantially, making it an expensive operation.

  For additional information about the KEY_BLOCK_SIZE option, see Table Options.

- Setting persistent table statistics options

```
ALTER TABLE tbl_name STATS_PERSISTENT=0, STATS_SAMPLE_PAGES=20, STATS_AUTO_RECALC=1, ALGORITHM=INPLACE, LOCK=NONE;
```

  Only modifies table metadata.

  Persistent statistics include STATS_PERSISTENT, STATS_AUTO_RECALC, and STATS_SAMPLE_PAGES. For more information, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

- Specifying a character set

```
ALTER TABLE tbl_name CHARACTER SET = charset_name, ALGORITHM=INPLACE, LOCK=NONE;
```

  Rebuilds the table if the new character encoding is different.

- Converting a character set

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name, ALGORITHM=COPY;
```

Rebuilds the table if the new character encoding is different.

- Optimizing a table

```
OPTIMIZE TABLE tbl_name;
```

In-place operation is not supported for tables with `FULLTEXT` indexes. The operation uses the `INPLACE` algorithm, but `ALGORITHM` and `LOCK` syntax is not permitted.

- Rebuilding a table with the `FORCE` option

```
ALTER TABLE tbl_name FORCE, ALGORITHM=INPLACE, LOCK=NONE;
```

Uses `ALGORITHM=INPLACE` as of MySQL 5.6.17. `ALGORITHM=INPLACE` is not supported for tables with `FULLTEXT` indexes.

- Performing a "null" rebuild

```
ALTER TABLE tbl_name ENGINE=InnoDB, ALGORITHM=INPLACE, LOCK=NONE;
```

Uses `ALGORITHM=INPLACE` as of MySQL 5.6.17. `ALGORITHM=INPLACE` is not supported for tables with `FULLTEXT` indexes.

- Renaming a table

```
ALTER TABLE old_tbl_name RENAME TO new_tbl_name, ALGORITHM=INPLACE, LOCK=NONE;
```

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 13.1.33, “RENAME TABLE Statement”.) Privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.

Tablespace Operations

The following table provides an overview of online DDL support for tablespace operations. For details, see Syntax and Usage Notes.

Table 14.16 Online DDL Support for Tablespace Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Enabling or disabling file-per-table tablespace encryption	No	Yes	No	No

Syntax and Usage Notes

Enabling or disabling file-per-table tablespace encryption

```
ALTER TABLE tbl_name ENCRYPTION='Y', ALGORITHM=COPY;
```

Encryption is only supported for file-per-table tablespaces. For related information, see Section 14.14, “InnoDB Data-at-Rest Encryption”.

Partitioning Operations

With the exception of most `ALTER TABLE` partitioning clauses, online DDL operations for partitioned `InnoDB` tables follow the same rules that apply to regular `InnoDB` tables.

Most ALTER TABLE partitioning clauses do not go through the same internal online DDL API as regular non-partitioned InnoDB tables. As a result, online support for ALTER TABLE partitioning clauses varies.

The following table shows the online status for each ALTER TABLE partitioning statement. Regardless of the online DDL API that is used, MySQL attempts to minimize data copying and locking where possible.

ALTER TABLE partitioning options that use ALGORITHM=COPY or that only permit "ALGORITHM=DEFAULT, LOCK=DEFAULT", repartition the table using the COPY algorithm. In other words, a new partitioned table is created with the new partitioning scheme. The newly created table includes any changes applied by the ALTER TABLE statement, and table data is copied into the new table structure.

Table 14.17 Online DDL Support for Partitioning Operations

Partitioning Clause	In Place	Permits DML	Notes
PARTITION BY	No	No	Permits ALGORITHM=COPY, LOCK={DEFAULT|SHARED|EXCLUSIVE}
ADD PARTITION	No	No	Only permits ALGORITHM=DEFAULT, LOCK=DEFAULT. Does not copy existing data for tables partitioned by RANGE or LIST. Concurrent queries are permitted for tables partitioned by HASH or LIST. MySQL copies the data while holding a shared lock.
DROP PARTITION	No	No	Only permits ALGORITHM=DEFAULT, LOCK=DEFAULT. Does not copy existing data for tables partitioned by RANGE or LIST.
DISCARD PARTITION	No	No	Only permits ALGORITHM=DEFAULT, LOCK=DEFAULT
IMPORT PARTITION	No	No	Only permits ALGORITHM=DEFAULT, LOCK=DEFAULT
TRUNCATE PARTITION	Yes	Yes	Does not copy existing data. It merely deletes rows; it does not alter the definition of the table itself, or of any of its partitions.

Partitioning Clause	In Place	Permits DML	Notes
`COALESCE PARTITION`	No	No	Only permits `ALGORITHM=DEFAULT`, `LOCK=DEFAULT`. Concurrent queries are permitted for tables partitioned by `HASH` or `LIST`, as MySQL copies the data while holding a shared lock.
`REORGANIZE PARTITION`	No	No	Only permits `ALGORITHM=DEFAULT`, `LOCK=DEFAULT`. Concurrent queries are permitted for tables partitioned by `LINEAR HASH` or `LIST`. MySQL copies data from affected partitions while holding a shared metadata lock.
`EXCHANGE PARTITION`	Yes	Yes	
`ANALYZE PARTITION`	Yes	Yes	
`CHECK PARTITION`	Yes	Yes	
`OPTIMIZE PARTITION`	No	No	`ALGORITHM` and `LOCK` clauses are ignored. Rebuilds the entire table. See Section 22.3.4, “Maintenance of Partitions”.
`REBUILD PARTITION`	No	No	Only permits `ALGORITHM=DEFAULT`, `LOCK=DEFAULT`. Concurrent queries are permitted for tables partitioned by `LINEAR HASH` or `LIST`. MySQL copies data from affected partitions while holding a shared metadata lock.
`REPAIR PARTITION`	Yes	Yes	
`REMOVE PARTITIONING`	No	No	Permits `ALGORITHM=COPY`, `LOCK={DEFAULT|SHARED|EXCLUSIVE}`

Non-partitioning online `ALTER TABLE` operations on partitioned tables follow the same rules that apply to regular tables. However, `ALTER TABLE` performs online operations on each table partition, which causes increased demand on system resources due to operations being performed on multiple partitions.

For additional information about ALTER TABLE partitioning clauses, see Partitioning Options, and Section 13.1.8.1, “ALTER TABLE Partition Operations”. For information about partitioning in general, see Chapter 22, *Partitioning*.