15.1.20.5 FOREIGN KEY Constraints

MySQL supports foreign keys, which permit cross-referencing related data across tables, and foreign key constraints, which help keep the related data consistent.

A foreign key relationship involves a parent table that holds the initial column values, and a child table with column values that reference the parent column values. A foreign key constraint is defined on the child table.

The essential syntax for a defining a foreign key constraint in a `CREATE TABLE` or `ALTER TABLE` statement includes the following:

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (col_name, ...)
    REFERENCES tbl_name (col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

Foreign key constraint usage is described under the following topics in this section:

- Identifiers
- Conditions and Restrictions
- Referential Actions
- Foreign Key Constraint Examples
- Adding Foreign Key Constraints
- Dropping Foreign Key Constraints
- Foreign Key Checks
- Locking
- Foreign Key Definitions and Metadata
- Foreign Key Errors

Identifiers

Foreign key constraint naming is governed by the following rules:

- The `CONSTRAINT symbol` value is used, if defined.
- If the `CONSTRAINT symbol` clause is not defined, or a symbol is not included following the `CONSTRAINT` keyword, a constraint name name is generated automatically.

  Prior to MySQL 8.0.16, if the `CONSTRAINT symbol` clause was not defined, or a symbol was not included following the `CONSTRAINT` keyword, both `InnoDB` and `NDB` storage engines would use the `FOREIGN_KEY index_name` if defined. In MySQL 8.0.16 and higher, the `FOREIGN_KEY index_name` is ignored.
- The `CONSTRAINT symbol` value, if defined, must be unique in the database. A duplicate `symbol` results in an error similar to: `ERROR 1005 (HY000): Can't create table 'test.fk1' (errno: 121).`
- NDB Cluster stores foreign names using the same lettercase with which they are created. Prior to version 8.0.20, when processing `SELECT` and other SQL statements, `NDB` compared the names of foreign keys in such statements with the names as stored in a case-sensitive fashion when `lower_case_table_names` was equal to 0. In NDB 8.0.20 and later, this value no longer has any effect on how such comparisons are made, and they are always done without regard to lettercase. (Bug #30512043)

Table and column identifiers in a `FOREIGN KEY ... REFERENCES` clause can be quoted within backticks (`` ` ``). Alternatively, double quotation marks (`"`) can be used if the `ANSI_QUOTES` SQL mode is enabled. The `lower_case_table_names` system variable setting is also taken into account.

Conditions and Restrictions

Foreign key constraints are subject to the following conditions and restrictions:

- Parent and child tables must use the same storage engine, and they cannot be defined as temporary tables.
- Creating a foreign key constraint requires the `REFERENCES` privilege on the parent table.
- Corresponding columns in the foreign key and the referenced key must have similar data types. *The size and sign of fixed precision types such as `INTEGER` and `DECIMAL` must be the same*. The length of string types need not be the same. For nonbinary (character) string columns, the character set and collation must be the same.
- MySQL supports foreign key references between one column and another within a table. (A column cannot have a foreign key reference to itself.) In these cases, a “child table record” refers to a dependent record within the same table.

- MySQL requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist. This index might be silently dropped later if you create another index that can be used to enforce the foreign key constraint. ***index_name***, if given, is used as described previously.

- `InnoDB` permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are the *first* columns in the same order. Hidden columns that `InnoDB` adds to an index are also considered (see Section 17.6.2.1, “Clustered and Secondary Indexes”).

  `NDB` requires an explicit unique key (or primary key) on any column referenced as a foreign key. `InnoDB` does not, which is an extension of standard SQL.

- Index prefixes on foreign key columns are not supported. Consequently, `BLOB` and `TEXT` columns cannot be included in a foreign key because indexes on those columns must always include a prefix length.

- `InnoDB` does not currently support foreign keys for tables with user-defined partitioning. This includes both parent and child tables.

  This restriction does not apply for `NDB` tables that are partitioned by `KEY` or `LINEAR KEY` (the only user partitioning types supported by the `NDB` storage engine); these may have foreign key references or be the targets of such references.

- A table in a foreign key relationship cannot be altered to use another storage engine. To change the storage engine, you must drop any foreign key constraints first.

- A foreign key constraint cannot reference a virtual generated column.

For information about how the MySQL implementation of foreign key constraints differs from the SQL standard, see Section 1.6.2.3, “FOREIGN KEY Constraint Differences”.

Referential Actions

When an `UPDATE` or `DELETE` operation affects a key value in the parent table that has matching rows in the child table, the result depends on the *referential action* specified by `ON UPDATE` and `ON DELETE` subclauses of the `FOREIGN KEY` clause. Referential actions include:

- `CASCADE`: Delete or update the row from the parent table and automatically delete or update the matching rows in the child table. Both `ON DELETE CASCADE` and `ON UPDATE CASCADE` are supported. Between two tables, do not define several `ON UPDATE CASCADE` clauses that act on the same column in the parent table or in the child table.

If a FOREIGN KEY clause is defined on both tables in a foreign key relationship, making both tables a parent and child, an ON UPDATE CASCADE or ON DELETE CASCADE subclause defined for one FOREIGN KEY clause must be defined for the other in order for cascading operations to succeed. If an ON UPDATE CASCADE or ON DELETE CASCADE subclause is only defined for one FOREIGN KEY clause, cascading operations fail with an error.

> **Note**
>
> Cascaded foreign key actions do not activate triggers.

- SET NULL: Delete or update the row from the parent table and set the foreign key column or columns in the child table to NULL. Both ON DELETE SET NULL and ON UPDATE SET NULL clauses are supported.

  If you specify a SET NULL action, *make sure that you have not declared the columns in the child table as NOT NULL*.

- RESTRICT: Rejects the delete or update operation for the parent table. Specifying RESTRICT (or NO ACTION) is the same as omitting the ON DELETE or ON UPDATE clause.

- NO ACTION: A keyword from standard SQL. For InnoDB, this is equivalent to RESTRICT; the delete or update operation for the parent table is immediately rejected if there is a related foreign key value in the referenced table. NDB supports deferred checks, and NO ACTION specifies a deferred check; when this is used, constraint checks are not performed until commit time. Note that for NDB tables, this causes all foreign key checks made for both parent and child tables to be deferred.

- SET DEFAULT: This action is recognized by the MySQL parser, but both InnoDB and NDB reject table definitions containing ON DELETE SET DEFAULT or ON UPDATE SET DEFAULT clauses.

For storage engines that support foreign keys, MySQL rejects any INSERT or UPDATE operation that attempts to create a foreign key value in a child table if there is no matching candidate key value in the parent table.

For an ON DELETE or ON UPDATE that is not specified, the default action is always NO ACTION.

As the default, an ON DELETE NO ACTION or ON UPDATE NO ACTION clause that is specified explicitly does not appear in SHOW CREATE TABLE output or in tables dumped with **mysqldump**. RESTRICT, which is an equivalent non-default keyword, appears in SHOW CREATE TABLE output and in tables dumped with **mysqldump**.

For NDB tables, ON UPDATE CASCADE is not supported where the reference is to the parent table's primary key.

As of NDB 8.0.16: For NDB tables, ON DELETE CASCADE is not supported where the child table contains one or more columns of any of the TEXT or BLOB types. (Bug #89511, Bug #27484882)

InnoDB performs cascading operations using a depth-first search algorithm on the records of the index that corresponds to the foreign key constraint.

A foreign key constraint on a stored generated column cannot use CASCADE, SET NULL, or SET DEFAULT as ON UPDATE referential actions, nor can it use SET NULL or SET DEFAULT as ON DELETE referential actions.

A foreign key constraint on the base column of a stored generated column cannot use CASCADE, SET NULL, or SET DEFAULT as ON UPDATE or ON DELETE referential actions.

Foreign Key Constraint Examples

This simple example relates parent and child tables through a single-column foreign key:

```
CREATE TABLE parent (
    id INT NOT NULL,
    PRIMARY KEY (id)
) ENGINE=INNODB;

CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
        ON DELETE CASCADE
) ENGINE=INNODB;
```

This is a more complex example in which a product_order table has foreign keys for two other tables. One foreign key references a two-column index in the product table. The other references a single-column index in the customer table:

```
CREATE TABLE product (
    category INT NOT NULL, id INT NOT NULL,
    price DECIMAL,
    PRIMARY KEY(category, id)
)   ENGINE=INNODB;

CREATE TABLE customer (
    id INT NOT NULL,
    PRIMARY KEY (id)
)   ENGINE=INNODB;
```

```
CREATE TABLE product_order (
    no INT NOT NULL AUTO_INCREMENT,
    product_category INT NOT NULL,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,

    PRIMARY KEY(no),
    INDEX (product_category, product_id),
    INDEX (customer_id),

    FOREIGN KEY (product_category, product_id)
      REFERENCES product(category, id)
      ON UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY (customer_id)
      REFERENCES customer(id)
)   ENGINE=INNODB;
```

Adding Foreign Key Constraints

You can add a foreign key constraint to an existing table using the following `ALTER TABLE` syntax:

```
ALTER TABLE tbl_name
    ADD [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (col_name, ...)
    REFERENCES tbl_name (col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

The foreign key can be self referential (referring to the same table). When you add a foreign key constraint to a table using `ALTER TABLE`, *remember to first create an index on the column(s) referenced by the foreign key.*

Dropping Foreign Key Constraints

You can drop a foreign key constraint using the following `ALTER TABLE` syntax:

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

If the `FOREIGN KEY` clause defined a `CONSTRAINT` name when you created the constraint, you can refer to that name to drop the foreign key constraint. Otherwise, a constraint name was generated internally, and you must use that value. To determine the foreign key constraint name, use `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int DEFAULT NULL,
  `parent_id` int DEFAULT NULL,
  KEY `par_ind` (`parent_id`),
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
  REFERENCES `parent` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

mysql> ALTER TABLE child DROP FOREIGN KEY `child_ibfk_1`;
```

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

Foreign Key Checks

In MySQL, InnoDB and NDB tables support checking of foreign key constraints. Foreign key checking is controlled by the `foreign_key_checks` variable, which is enabled by default. Typically, you leave this variable enabled during normal operation to enforce referential integrity. The `foreign_key_checks` variable has the same effect on `NDB` tables as it does for `InnoDB` tables.

The `foreign_key_checks` variable is dynamic and supports both global and session scopes. For information about using system variables, see Section 7.1.9, “Using System Variables”.

Disabling foreign key checking is useful when:

- Dropping a table that is referenced by a foreign key constraint. A referenced table can only be dropped after `foreign_key_checks` is disabled. When you drop a table, constraints defined on the table are also dropped.
- Reloading tables in different order than required by their foreign key relationships. For example, **mysqldump** produces correct definitions of tables in the dump file, including foreign key constraints for child tables. To make it easier to reload dump files for tables with foreign key relationships, **mysqldump** automatically includes a statement in the dump output that disables `foreign_key_checks`. This enables you to import the tables in any order in case the dump file contains tables that are not correctly ordered for foreign keys. Disabling `foreign_key_checks` also speeds up the import operation by avoiding foreign key checks.
- Executing `LOAD DATA` operations, to avoid foreign key checking.
- Performing an `ALTER TABLE` operation on a table that has a foreign key relationship.

When `foreign_key_checks` is disabled, foreign key constraints are ignored, with the following exceptions:

- Recreating a table that was previously dropped returns an error if the table definition does not conform to the foreign key constraints that reference the table. The table must have the correct column names and types. It must also have indexes on the referenced keys. If these requirements are not satisfied, MySQL returns Error 1005 that refers to errno: 150 in the error message, which means that a foreign key constraint was not correctly formed.
- Altering a table returns an error (errno: 150) if a foreign key definition is incorrectly formed for the altered table.
- Dropping an index required by a foreign key constraint. The foreign key constraint must be removed before dropping the index.
- Creating a foreign key constraint where a column references a nonmatching column type.

Disabling `foreign_key_checks` has these additional implications:

- It is permitted to drop a database that contains tables with foreign keys that are referenced by tables outside the database.
- It is permitted to drop a table with foreign keys referenced by other tables.
- Enabling `foreign_key_checks` does not trigger a scan of table data, which means that rows added to a table while `foreign_key_checks` is disabled are not checked for consistency when `foreign_key_checks` is re-enabled.

Locking

MySQL extends metadata locks, as necessary, to tables that are related by a foreign key constraint. Extending metadata locks prevents conflicting DML and DDL operations from executing concurrently on related tables. This feature also enables updates to foreign key metadata when a parent table is modified. In earlier MySQL releases, foreign key metadata, which is owned by the child table, could not be updated safely.

If a table is locked explicitly with `LOCK TABLES`, any tables related by a foreign key constraint are opened and locked implicitly. For foreign key checks, a shared read-only lock (`LOCK TABLES READ`) is taken on related tables. For cascading updates, a shared-nothing write lock (`LOCK TABLES WRITE`) is taken on related tables that are involved in the operation.

Foreign Key Definitions and Metadata

To view a foreign key definition, use `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int DEFAULT NULL,
  `parent_id` int DEFAULT NULL,
  KEY `par_ind` (`parent_id`),
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
  REFERENCES `parent` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

You can obtain information about foreign keys from the Information Schema `KEY_COLUMN_USAGE` table. An example of a query against this table is shown here:

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME
       FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
       WHERE REFERENCED_TABLE_SCHEMA IS NOT NULL;
+--------------+------------+-------------+-----------------+
| TABLE_SCHEMA | TABLE_NAME | COLUMN_NAME | CONSTRAINT_NAME |
+--------------+------------+-------------+-----------------+
| test         | child      | parent_id   | child_ibfk_1    |
+--------------+------------+-------------+-----------------+
```

You can obtain information specific to `InnoDB` foreign keys from the `INNODB_FOREIGN` and `INNODB_FOREIGN_COLS` tables. Example queries are show here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN \G
*************************** 1. row ***************************
      ID: test/child_ibfk_1
FOR_NAME: test/child
REF_NAME: test/parent
  N_COLS: 1
    TYPE: 1

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN_COLS \G
*************************** 1. row ***************************
          ID: test/child_ibfk_1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
         POS: 0
```

Foreign Key Errors

In the event of a foreign key error involving `InnoDB` tables (usually Error 150 in the MySQL Server), information about the latest foreign key error can be obtained by checking `SHOW ENGINE INNODB`

`STATUS` output.

```
mysql> SHOW ENGINE INNODB STATUS\G
...
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2018-04-12 14:57:24 0x7f97a9c9170O Transaction:
TRANSACTION 7717, ACTIVE 0 sec inserting
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1136, 3 row lock(s), undo log entries 3
MySQL thread id 8, OS thread handle 140289365317376, query id 14 localhost root up
INSERT INTO child VALUES (NULL, 1), (NULL, 2), (NULL, 3), (NULL, 4), (NULL, 5), (N
Foreign key constraint fails for table `test`.`child`:
,
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent` (`id`) C
  CASCADE ON UPDATE CASCADE
Trying to add in child table, in index par_ind tuple:
DATA TUPLE: 2 fields;
 0: len 4; hex 80000003; asc     ;;
 1: len 4; hex 80000003; asc     ;;

But in parent table `test`.`parent`, in index PRIMARY,
the closest match we can find is record:
PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000004; asc     ;;
 1: len 6; hex 000000001e19; asc       ;;
 2: len 7; hex 81000001110137; asc       7;;
...
```

> **Warning**
>
> If a user has table-level privileges for all parent tables, `ER_NO_REFERENCED_ROW_2` and `ER_ROW_IS_REFERENCED_2` error messages for foreign key operations expose information about parent tables. If a user does not have table-level privileges for all parent tables, more generic error messages are displayed instead (`ER_NO_REFERENCED_ROW` and `ER_ROW_IS_REFERENCED`).
>
> An exception is that, for stored programs defined to execute with `DEFINER` privileges, the user against which privileges are assessed is the user in the program `DEFINER` clause, not the invoking user. If that user has table-level parent table privileges, parent table information is still displayed. In this case, it is the responsibility of the stored program creator to hide the information by including appropriate condition handlers.