

MySQL Foreign Key

The foreign key is used to link one or more than one table together. It is also known as the **referencing** key. A foreign key matches the primary key field of another table. It means a foreign key field in one table refers to the primary key field of the other table. It identifies each row of another table uniquely that maintains the **referential integrity** in MySQL.

A foreign key makes it possible to create a parent-child relationship with the tables. In this relationship, the parent table holds the initial column values, and column values of child table reference the parent column values. MySQL allows us to define a foreign key constraint on the child table.

MySQL defines the foreign key in two ways:

1. Using CREATE TABLE Statement
2. Using ALTER TABLE Statement

Syntax

Following are the basic syntax used for defining a foreign key using CREATE TABLE OR ALTER TABLE statement in the MySQL:

```
CONSTRAINT  
FOREIGN KEY  
REFERENCES  
ON DELETE
```

In the above syntax, we can see the following parameters:

constraint_name: It specifies the name of the foreign key constraint. If we have not provided the constraint name, MySQL generates its name automatically.

col_name: It is the names of the column that we are going to make foreign key.

parent_tbl_name: It specifies the name of a parent table followed by column names that reference the foreign key columns.

Refrence_option: It is used to ensure how foreign key maintains referential integrity using ON DELETE and ON UPDATE clause between parent and child table.

MySQL contains **five** different referential options, which are given below:

CASCADE: It is used when we delete or update any row from the parent table, the values of the matching rows in the child table will be deleted or updated automatically.

SET NULL: It is used when we delete or update any row from the parent table, the values of the foreign key columns in the child table are set to NULL.

RESTRICT: It is used when we delete or update any row from the parent table that has a matching row in the reference(child) table, MySQL does not allow to delete or update rows in the parent table.

NO ACTION: It is similar to RESTRICT. But it has one difference that it checks referential integrity after trying to modify the table.

SET DEFAULT: The MySQL parser recognizes this action. However, the InnoDB and NDB tables both rejected this action.

NOTE: MySQL mainly provides full support to CASCADE, RESTRICT, and SET NULL actions. If we have not specified the ON DELETE and ON UPDATE clause, MySQL takes default action RESTRICT.

Foreign Key Example

Let us understand how foreign key works in MySQL. So first, we are going to create a database named "**mysqltestdb**" and start using it with the command below:

Next, we need to create two tables named "**customer**" and "**contact**" using the below statement:

Table: customer

```
CREATE TABLE customer
(
    id INT NOT NULL,
    Name varchar(50) NOT NULL,
    Address varchar(100) NOT NULL,
    PRIMARY KEY (id)
);
```

Table: contact

```
CREATE TABLE contact
(
    id INT NOT NULL,
    CustomerId INT NOT NULL,
    Name varchar(50) NOT NULL,
    Address varchar(100) NOT NULL,
    INDEX (id),
    CONSTRAINT fk_contact_customer FOREIGN KEY (CustomerId)
    REFERENCES customer (id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

Table Structure Verification

Here, we are going to see how our database structure looks like using the following queries:

```
SELECT * FROM customer;
SELECT * FROM contact;
```

We will get the structure as below:

```
MySQL 8.0 Command Line Client

mysql> SHOW TABLES;
+-----+
| Tables_in_mysqltestdb |
+-----+
| contact                |
| customer                |
+-----+
2 rows in set (0.00 sec)

mysql> DESCRIBE customer;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID    | int           | NO   | PRI | NULL    | auto_increment |
| Name  | varchar(50)   | NO   |     | NULL    |                |
| City  | varchar(50)   | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> DESCRIBE contact;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID             | int           | YES  |     | NULL    |                |
| Customer_Id    | int           | YES  | MUL | NULL    |                |
| Customer_Info  | varchar(50)   | NO   |     | NULL    |                |
| Type           | varchar(50)   | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

In the above output, we can see that the **PRI** in the key column of the customer table tells that this field is the primary index value. Next, the **MUL** in the key column of the contact value tells that the **Customer_Id** field can store multiple rows with the same value.

Insert Data to the Table

Now, we have to insert the records into both tables. Execute this statement to insert data into table customer:

```
INSERT INTO      Name      VALUES
'Joseph' 'California'
'Mary' 'NewYork'
'John' 'Alaska'
```

After insertion, execute the SELECT TABLE command to check the customer table data as below:

```
MySQL 8.0 Command Line Client
mysql> SELECT * FROM customer;
+-----+-----+-----+
| ID | Name | City |
+-----+-----+-----+
| 1 | Joseph | California |
| 2 | Mary | NewYork |
| 3 | John | Alaska |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

Execute the below insert statement to add data into a table contact:

```
INSERT INTO                               VALUES
'Joseph@javatpoint.com' 'email'
'121-121-121' 'work'
'123-123-123' 'home'
'Mary@javatpoint.com' 'email'
'Mary@javatpoint.com' 'email'
'212-212-212' 'work'
'John@javatpoint.com' 'email'
'313-313-313' 'home'
```

Our contact table looks like as below:

```
MySQL 8.0 Command Line Client
mysql> SELECT * FROM contact;
+-----+-----+-----+-----+
| ID | Customer_Id | Customer_Info | Type |
+-----+-----+-----+-----+
| NULL | 1 | Joseph@javatpoint.com | email |
| NULL | 1 | 121-121-121 | work |
| NULL | 1 | 123-123-123 | home |
| NULL | 2 | Mary@javatpoint.com | email |
| NULL | 2 | Mary@javatpoint.com | email |
| NULL | 2 | 212-212-212 | work |
| NULL | 3 | John@javatpoint.com | email |
| NULL | 3 | 313-313-313 | home |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)
```

Now, let's see how foreign keys in MySQL preserve data integrity.

So here, we are going to delete the referential data that removes records from both tables. We have defined the foreign key in the contact table as:

FOREIGN KEY **REFERENCES**

ON DELETE CASCADE

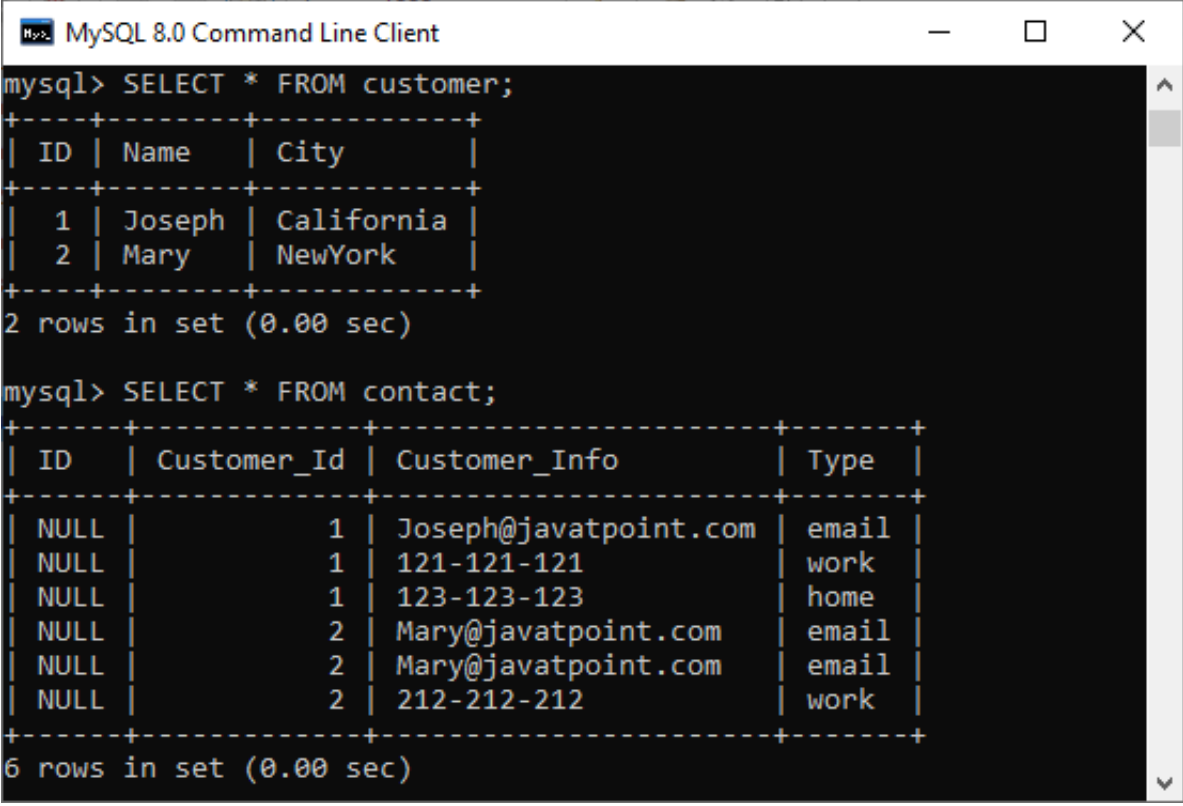
ON UPDATE CASCADE

It means if we delete any customer record from the customer table, then the related records in the contact table should also be deleted. And the ON UPDATE CASCADE will updates automatically on the parent table to referenced fields in the child table(Here, it is Customer_Id).

Execute this statement that deletes a record from the table whose name is **JOHN**.

DELETE FROM **WHERE Name 'John'**

Again, if we look at our tables, we can see that both tables were changed. It means the fields with name JOHN will be removed entirely from both tables.



```
mysql> SELECT * FROM customer;
+----+-----+-----+
| ID | Name  | City    |
+----+-----+-----+
| 1  | Joseph | California |
| 2  | Mary  | NewYork  |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM contact;
+----+-----+-----+-----+-----+
| ID   | Customer_Id | Customer_Info          | Type |
+----+-----+-----+-----+-----+
| NULL | 1           | Joseph@javatpoint.com | email |
| NULL | 1           | 121-121-121           | work  |
| NULL | 1           | 123-123-123           | home  |
| NULL | 2           | Mary@javatpoint.com   | email |
| NULL | 2           | Mary@javatpoint.com   | email |
| NULL | 2           | 212-212-212           | work  |
+----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Now, test the **ON UPDATE CASCADE**. Here, we are going to update the Customer_Id of **Mary** in the contact table as:

UPDATE **SET** **WHERE Name 'Mary'**

Again, if we look at our tables, we can see that both tables were changed with Customer_Id of Mary=3.

```
MySQL 8.0 Command Line Client
mysql> SELECT * FROM customer;
+-----+-----+-----+
| ID | Name | City |
+-----+-----+-----+
| 1 | Joseph | California |
| 3 | Mary | NewYork |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM contact;
+-----+-----+-----+-----+
| ID | Customer_Id | Customer_Info | Type |
+-----+-----+-----+-----+
| NULL | 1 | Joseph@javatpoint.com | email |
| NULL | 1 | 121-121-121 | work |
| NULL | 1 | 123-123-123 | home |
| NULL | 3 | Mary@javatpoint.com | email |
| NULL | 3 | Mary@javatpoint.com | email |
| NULL | 3 | 212-212-212 | work |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Foreign Key example using SET NULL action

Here, we are going to understand how the SET NULL action works with a foreign key. First, we have to create two table named **Persons** and **Contacts**, as shown below:

Table: Persons

```
CREATE TABLE
  INT NOT NULL
  Name varchar NOT NULL
  varchar NOT NULL
  PRIMARY KEY
```

Table: Customers

```
CREATE TABLE
  INT
  INT
  varchar NOT NULL
  varchar NOT NULL
  INDEX
```

```
CONSTRAINT FOREIGN KEY
```

```
REFERENCES
```

```
ON DELETE SET NULL
```

```
ON UPDATE SET NULL
```

Next, we need to insert the data into both tables using the following statement:

```
INSERT INTO      Name      VALUES
```

```
'Joseph' 'Texas'
```

```
'Mary' 'Arizona'
```

```
'Peter' 'Alaska'
```

```
INSERT INTO      VALUES
```

```
'joseph@javatpoint.com' 'email'
```

```
'121-121-121' 'work'
```

```
'mary@javatpoint.com' 'email'
```

```
'212-212-212' 'work'
```

```
'peter@javatpoint.com' 'email'
```

```
'313-313-313' 'home'
```

Now, update the ID of the "Persons" table:

```
UPDATE
```

```
SET
```

```
WHERE
```

Finally, verify the update usi




```
MySQL 8.0 Command Line Client
mysql> SELECT * FROM Persons;
+-----+-----+-----+
| ID | Name | City |
+-----+-----+-----+
| 1 | Joseph | Texas |
| 2 | Mary | Arizona |
| 103 | Peter | Alaska |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Contacts;
+-----+-----+-----+-----+-----+
| ID | Person_Id | Info | Type |
+-----+-----+-----+-----+-----+
| NULL | 1 | joseph@javatpoint.com | email |
| NULL | 1 | 121-121-121 | work |
| NULL | 2 | mary@javatpoint.com | email |
| NULL | 2 | 212-212-212 | work |
| NULL | NULL | peter@javatpoint.com | email |
| NULL | NULL | 313-313-313 | home |
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

If we look at our tables, we can see that both tables were changed. The rows with a **Person_Id=3** in the Contacts table automatically set to **NULL** due to the ON UPDATE SET NULL action.

How to DROP Foreign Key

MySQL allows the ALTER TABLE statement to remove an existing foreign key from the table. The following syntax is used to drop a foreign key:

```
ALTER TABLE DROP FOREIGN KEY
```

Here, the **table_name** is the name of a table from where we are going to remove the foreign key. The **constraint_name** is the name of the foreign key that was added during the creation of a table.

If we have not known the name of an existing foreign key into the table, execute the following command:

```
CREATE TABLE
```

It will give the output as below where we can see that the table contact has one foreign key named **fk_customer** shown in the red rectangle.

Table	Create Table
contact	<pre>CREATE TABLE `contact` (`ID` int DEFAULT NULL, `Customer_Id` int DEFAULT NULL, `Customer_Info` varchar(50) NOT NULL, `Type` varchar(50) NOT NULL, KEY `par_ind` (`Customer_Id`), CONSTRAINT `fk_customer` FOREIGN KEY (`Customer_Id`) REFERENCES `customer` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci</pre>

Now, to delete this foreign key constraint from the contact table, execute the statement as below:

ALTER TABLE

DROP FOREIGN KEY

We can verify whether foreign key constraint removes or not, use the SHOW CREATE TABLE statement. It will give the output as below where we can see that the foreign key is no longer available in the table contact.

Table	Create Table
contact	<pre>CREATE TABLE `contact` (`ID` int DEFAULT NULL, `Customer_Id` int DEFAULT NULL, `Customer_Info` varchar(50) NOT NULL, `Type` varchar(50) NOT NULL, KEY `par_ind` (`Customer_Id`)) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci</pre>

Define Foreign Key Using ALTER TABLE Statement

This statement allows us to do the modification into the existing table. Sometimes there is a need to add a foreign key to the column of an existing table; then, this statement is used to add the foreign key for that column.

Syntax

Following are the syntax of the ALTER TABLE statement to add a foreign key in the existing table:

ALTER TABLE

ADD CONSTRAINT

FOREIGN KEY

REFERENCES

ON DELETE

ON UPDATE

When we add a foreign key using the ALTER TABLE statement, it is recommended to first create an **index** on the column(s), which is referenced by the foreign key.

Example

The following statement creates two tables, "**Person**" and "**Contact**", without having a foreign key column into the table definition.

Table: Person

```
CREATE TABLE
  INT NOT NULL
  Name varchar NOT NULL
  varchar NOT NULL
  PRIMARY KEY
```

Table: Contact

```
CREATE TABLE
  INT
  INT
  varchar NOT NULL
  varchar NOT NULL
```

After creating a table, if we want to add a foreign key to an existing table, we need to execute the ALTER TABLE statement as below:

```
ALTER TABLE      ADD INDEX
ALTER TABLE      ADD CONSTRAINT
FOREIGN KEY        REFERENCES          ON DELETE CASCADE ON UPDATE RESTRICT
```

Foreign Key Checks

MySQL has a special variable **foreign_key_checks** to control the foreign key checking into the tables. By default, it is enabled to enforce the referential integrity during the normal operation on the tables. This variable is dynamic in nature so that it supports global and session scopes both.

Sometimes there is a need for disabling the foreign key checking, which is very useful when:

- We drop a table that is a reference by the foreign key.

- We import data from a CSV file into a table. It speeds up the import operation.
- We use ALTER TABLE statement on that table which has a foreign key.
- We can execute load data operation into a table in any order to avoid foreign key checking.

The following statement allows us to **disable** foreign key checks:

```
SET
```

The following statement allows us to **enable** foreign key checks:

```
SET
```

← prev

next →

Help Others, Please Share



▶ Get Started

Learn Latest Tutorials

IoT Tutorial

IoT

GraphQL
Tutorial

GraphQL

Xampp
Tutorial

Xampp

Kivy Tutorial

Kivy

Automation
Anywhere
Tutorial

A. Anywhere

Ext.js Tutorial

Ext.js

UiPath
Tutorial

UiPath

Arduino
tutorial

Arduino

Digital
Electronics
tutorial

Digital E.

Google
Adwords
tutorial

Adwords

MySQL
tutorial

MySQL

Python
tutorial

Python

smartsheet

Smartsheet

affiliate
marketing

Affiliate M.

Preparation



Trending Technologies



	ML	DevOps
--	----	--------

B.Tech / MCA

DBMS tutorial DBMS	Data Structures tutorial DS	DAA tutorial DAA
Operating System tutorial OS	Computer Network tutorial C. Network	Compiler Design tutorial Compiler D.
Computer Organization and Architecture COA	Discrete Mathematics Tutorial D. Math.	Ethical Hacking Tutorial E. Hacking
Computer Graphics Tutorial C. Graphics	Software Engineering Tutorial Software E.	html tutorial Web Tech.
Cyber Security tutorial Cyber Sec.	Automata Tutorial Automata	C Language tutorial C
C++ tutorial C++	Java tutorial Java	.Net Framework tutorial .Net

Python
tutorial

Python

List of
Programs

Programs

Control
Systems
tutorial

Control S.

Data Mining
Tutorial

Data Mining