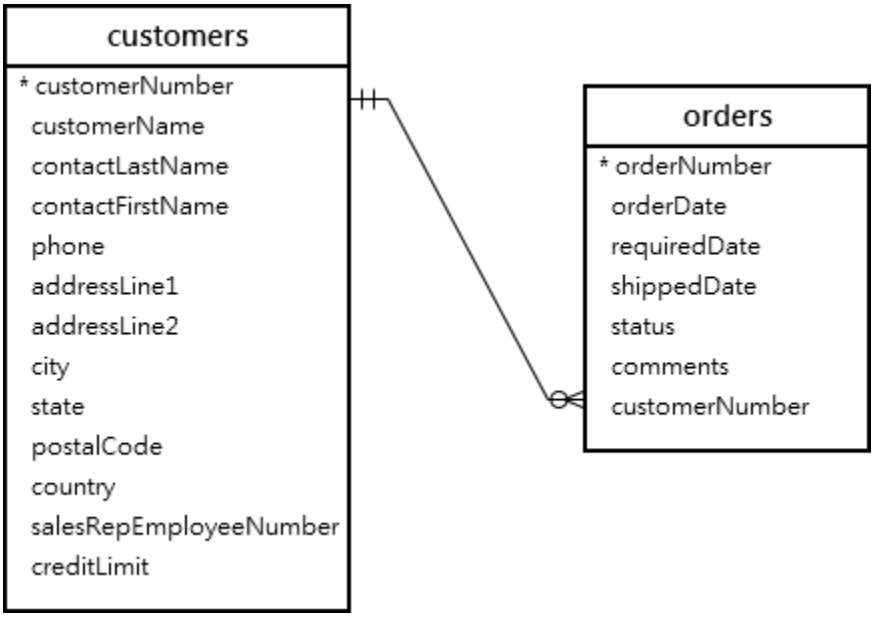
Introduction to MySQL foreign key

A foreign key is a column or group of columns in a table that links to a column or group of columns in another table. The foreign key places constraints on data in the related tables, which allows MySQL to maintain referential integrity.

Let's take a look at the following `customers` and `orders` tables from the sample database.



In this diagram, each customer can have zero or many orders and each order belongs to one customer.

The relationship between `customers` table and `orders` table is one-to-many. And this relationship is established by the foreign key in the `orders` table specified by the `customerNumber` column.

The `customerNumber` column in the `orders` table links to the `customerNumber` primary key column in the `customers` table.

The `customers` table is called the *parent table* or *referenced table*, and the `orders` table is known as the *child table* or *referencing table*.

Typically, the foreign key columns of the child table often refer to the primary key columns of the parent table.

A table can have more than one foreign key where each foreign key references to a primary key of the different parent tables.

Once a foreign key constraint is in place, the foreign key columns from the child table must have the corresponding row in the parent key columns of the parent table or values in these foreign key column must be `NULL` (see the `SET NULL` action example below).

For example, each row in the `orders` table has a `customerNumber` that exists in the `customerNumber` column of the `customers` table. Multiple rows in the `orders` table can have the same `customerNumber`.

MySQL `FOREIGN KEY` syntax

Here is the basic syntax of defining a foreign key constraint in the `CREATE TABLE` or `ALTER TABLE` statement:

```
[CONSTRAINT constraint_name]
FOREIGN KEY [foreign_key_name] (column_name, ...)
REFERENCES parent_table(colunm_name,...)
[ON DELETE reference_option]
[ON UPDATE reference_option]
```

In this syntax:

First, specify the name of foreign key constraint that you want to create after the `CONSTRAINT` keyword. If you omit the constraint name, MySQL automatically generates a name for the foreign key constraint.

Second, specify a list of comma-separated foreign key columns after the `FOREIGN KEY` keywords. The foreign key name is also optional and is generated automatically if you skip it.

Third, specify the parent table followed by a list of comma-separated columns to which the foreign key columns reference.

Finally, specify how foreign key maintains the referential integrity between the child and parent tables by using the `ON DELETE` and `ON UPDATE` clauses. The `reference_option` determines action which MySQL will take when values in the parent key columns are deleted (`ON DELETE`) or updated (`ON UPDATE`).

MySQL has five reference options: `CASCADE`, `SET NULL`, `NO ACTION`, `RESTRICT`, and `SET DEFAULT`.

- **`CASCADE:`** if a row from the parent table is deleted or updated, the values of the matching rows in the child table automatically deleted or updated.
- **`SET NULL:`** if a row from the parent table is deleted or updated, the values of the foreign key column (or columns) in the child table are set to `NULL`.
- **`RESTRICT:`** if a row from the parent table has a matching row in the child table, MySQL rejects deleting or updating rows in the parent table.
- **`NO ACTION:`** is the same as `RESTRICT`.

- **SET DEFAULT:** is recognized by the MySQL parser. However, this action is rejected by both InnoDB and NDB tables.

In fact, MySQL fully supports three actions: RESTRICT, CASCADE and SET NULL.

If you don't specify the `ON DELETE` and `ON UPDATE` clause, the default action is `RESTRICT`.

MySQL FOREIGN KEY examples

Let's create a new database called `fkdemo` for the demonstration.

```
CREATE DATABASE fkdemo;

USE fkdemo;
```

RESTRICT & NO ACTION actions

Inside the `fkdemo` database, create two tables `categories` and `products`:

```
CREATE TABLE categories(
    categoryId INT AUTO_INCREMENT PRIMARY KEY,
    categoryName VARCHAR(100) NOT NULL
) ENGINE=INNODB;

CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
        REFERENCES categories(categoryId)
) ENGINE=INNODB;
```

The `categoryId` in the `products` table is the foreign key column that refers to the `categoryId` column in the `categories` table.

Because we don't specify any `ON UPDATE` and `ON DELETE` clauses, the default action is `RESTRICT` for both update and delete operation.

The following steps illustrate the `RESTRICT` action.

1) Insert two rows into the `categories` table:

```
INSERT INTO categories(categoryName)
VALUES
    ('Smartphone'),
    ('Smartwatch');
```

2) Select data from the `categories` table:

```
SELECT * FROM categories;
```

categoryId	categoryName
1	Smartphone
2	Smartwatch

3) Insert a new row into the `products` table:

```
INSERT INTO products(productName, categoryId)
VALUES('iPhone',1);
```

It works because the `categoryId` 1 exists in the `categories` table.

4) Attempt to insert a new row into the `products` table with a `categoryId` value does not exist in the `categories` table:

```
INSERT INTO products(productName, categoryId)
VALUES('iPad',3);
```

MySQL issued the following error:

```
Error Code: 1452. Cannot add or update a child row: a foreign key constraint
fails (`fkdemo`.`products`, CONSTRAINT `fk_category` FOREIGN KEY
(`categoryId`) REFERENCES `categories` (`categoryId`) ON DELETE RESTRICT ON
UPDATE RESTRICT)
```

5) Update the value in the `categoryId` column in the `categories` table to `100`:

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

MySQL issued this error:

```
Error Code: 1451. Cannot delete or update a parent row: a foreign key
constraint fails (`fkdemo`.`products`, CONSTRAINT `fk_category` FOREIGN KEY
(`categoryId`) REFERENCES `categories` (`categoryId`) ON DELETE RESTRICT ON
UPDATE RESTRICT)
```

Because of the `RESTRICT` option, you cannot delete or update `categoryId 1` since it is referenced by the `productId 1` in the `products` table.

`CASCADE` action

These steps illustrate how `ON UPDATE CASCADE` and `ON DELETE CASCADE` actions work.

1) Drop the `products` table:

```
DROP TABLE products;
```

2) Create the `products` table with the `ON UPDATE CASCADE` and `ON DELETE CASCADE` options for the foreign key:

```
CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT NOT NULL,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
    REFERENCES categories(categoryId)
        ON UPDATE CASCADE
        ON DELETE CASCADE
) ENGINE=INNODB;
```

3) Insert four rows into the `products` table:

```
INSERT INTO products(productName, categoryId)
VALUES
    ('iPhone', 1),
    ('Galaxy Note',1),
    ('Apple Watch',2),
    ('Samsung Galary Watch',2);
```

4) Select data from the `products` table:

```
SELECT * FROM products;
```

productId	productName	categoryId
1	iPhone	1
2	Galaxy Note	1
3	Apple Watch	2
4	Samsung Galary Watch	2

5) Update `categoryId` 1 to 100 in the `categories` table:

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

6) Verify the update:

```
SELECT * FROM categories;
```

categoryId	categoryName
2	Smartwatch
100	Smartphone

7) Get data from the `products` table:

```
SELECT * FROM products;
```

	productId	productName	categoryId
▶	1	iPhone	100
	2	Galaxy Note	100
	3	Apple Watch	2
	4	Samsung Galary Watch	2

As you can see, two rows with value `1` in the `categoryId` column of the `products` table were automatically updated to `100` because of the `ON UPDATE CASCADE` action.

8) Delete `categoryId` 2 from the `categories` table:

```
DELETE FROM categories
WHERE categoryId = 2;
```

9) Verify the deletion:

```
SELECT * FROM categories;
```

	categoryId	categoryName
▶	100	Smartphone

10) Check the `products` table:

```
SELECT * FROM products;
```

	productId	productName	categoryId
▶	1	iPhone	100
	2	Galaxy Note	100

All products with `categoryId` 2 from the `products` table were automatically deleted because of the `ON DELETE CASCADE` action.

`SET NULL` action

These steps illustrate how the `ON UPDATE SET NULL` and `ON DELETE SET NULL` actions work.

1) Drop both `categories` and `products` tables:

```
DROP TABLE IF EXISTS categories;
DROP TABLE IF EXISTS products;
```

2) Create the `categories` and `products` tables:

```
CREATE TABLE categories(
    categoryId INT AUTO_INCREMENT PRIMARY KEY,
    categoryName VARCHAR(100) NOT NULL
)ENGINE=INNODB;
```

```
CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
        REFERENCES categories(categoryId)
        ON UPDATE SET NULL
        ON DELETE SET NULL
)ENGINE=INNODB;
```

The foreign key in the `products` table changed to `ON UPDATE SET NULL` and `ON DELETE SET NULL` options.

3) Insert rows into the `categories` table:

```
INSERT INTO categories(categoryName)
VALUES
    ('Smartphone'),
    ('Smartwatch');
```

4) Insert rows into the `products` table:

```
INSERT INTO products(productName, categoryId)
VALUES
    ('iPhone', 1),
    ('Galaxy Note',1),
    ('Apple Watch',2),
    ('Samsung Galary Watch',2);
```

5) Update `categoryId` from 1 to 100 in the `categories` table:

```
UPDATE categories
SET categoryId = 100
WHERE categoryId = 1;
```

6) Verify the update:

```
SELECT * FROM categories;
```

	categoryId	categoryName
▶	2	Smartwatch
	100	Smartphone

7) Select data from the `products` table:

	productId	productName	categoryId
▶	1	iPhone	NULL
	2	Galaxy Note	NULL
	3	Apple Watch	2
	4	Samsung Galary Watch	2

The rows with the `categoryId` 1 in the `products` table were automatically set to `NULL` due to the `ON UPDATE SET NULL` action.

8) Delete the `categoryId` 2 from the `categories` table:

```
DELETE FROM categories
WHERE categoryId = 2;
```

9) Check the `products` table:

```
SELECT * FROM products;
```

	productId	productName	categoryId
▶	1	iPhone	NULL
	2	Galaxy Note	NULL
	3	Apple Watch	NULL
	4	Samsung Galary Watch	NULL

The values in the `categoryId` column of the rows with `categoryId` 2 in the `products` table were automatically set to `NULL` due to the `ON DELETE SET NULL` action.

Drop MySQL foreign key constraints

To drop a foreign key constraint, you use the `ALTER TABLE` statement:

```
ALTER TABLE table_name
DROP FOREIGN KEY constraint_name;
```

In this syntax:

- First, specify the name of the table from which you want to drop the foreign key after the `ALTER TABLE` keywords.
- Second, specify the constraint name after the `DROP FOREIGN KEY` keywords.

Notice that `constraint_name` is the name of the foreign key constraint specified when you created or added the foreign key constraint to the table.

To obtain the generated constraint name of a table, you use the `SHOW CREATE TABLE` statement:

```
SHOW CREATE TABLE table_name;
```

For example, to see the foreign keys of the `products` table, you use the following statement:

```
SHOW CREATE TABLE products;
```

The following is the output of the statement:

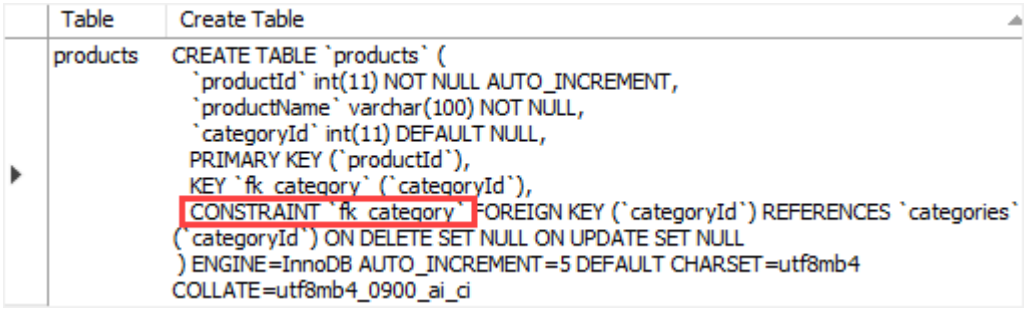


Table	Create Table
products	CREATE TABLE `products` (`productId` int(11) NOT NULL AUTO_INCREMENT, `productName` varchar(100) NOT NULL, `categoryId` int(11) DEFAULT NULL, PRIMARY KEY (`productId`), KEY `fk_category` (`categoryId`), CONSTRAINT `fk_category` FOREIGN KEY (`categoryId`) REFERENCES `categories` (`categoryId`) ON DELETE SET NULL ON UPDATE SET NULL) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

As you can see clearly from the output, the table `products` table has one foreign key constraint: `fk_category`

And this statement drops the foreign key constraint of the `products` table:

```
ALTER TABLE products
DROP FOREIGN KEY fk_category;
```

To ensure that the foreign key constraint has been dropped, you can view the structure of the products table:

```
SHOW CREATE TABLE products;
```

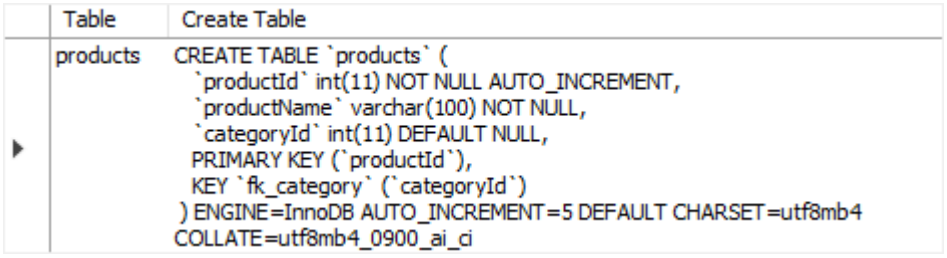


Table	Create Table
products	CREATE TABLE `products` (`productId` int(11) NOT NULL AUTO_INCREMENT, `productName` varchar(100) NOT NULL, `categoryId` int(11) DEFAULT NULL, PRIMARY KEY (`productId`), KEY `fk_category` (`categoryId`)) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci