# What Problems Can Foreign Keys Solve?

Congratulations on making it to the more advanced chapters!

In the first chapter, the concept `FOREIGN KEY`, which we have mentioned in the previous chapters, will be interpreted. In fact, a foreign key is the column of binding external data such as `user_id`, `order_id` that we set in the past.

In the past, we recorded data with relative columns and used our brains to determine whether they were related. But in practice, we can actually set a `CONSTRAINT` for the foreign key through the SQL to help us make automatic checks and protect data security.

What is data security? Let’s imagine a situation where a customer has a large amount of order data, but one day an engineer deletes the customer's order data by mistake. So if someone needs to inquire about the customer's order history in the future, they won't know that the deleted orders belong to this customer. This is a disaster in business.

Fortunately, through `FOREIGN KEY`, we can automatically and effectively avoid such problems. Let's see how to use it!

# Sample table

users

| id | name | age |
| --- | --- | --- |
| 1 | John | 40 |
| 2 | May | 30 |
| 3 | Jack | 22 |

orders

| id | user_id | note |
| --- | --- | --- |

| | | |
|---|---|---|
| 1 | 1 | some information |
| 2 | 2 | some comments |
| 3 | 2 | no comments |
| 4 | 3 | more comments |

To help solidify the new concepts, you can follow along by editing the provided table below:

## Schema SQL

```
CREATE SCHEMA `new_schema` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;


CREATE TABLE `new_schema`.`users` (
  `id` INT NOT NULL AUTO_INCREMENT COMMENT 'This is the primary index',
  `name` VARCHAR(45) NOT NULL DEFAULT 'N/A',
  `age` INT NULL,
  PRIMARY KEY (`id`)
);


INSERT INTO `new_schema`.`users` (`id`, `name`, `age`) VALUES
  (1, 'John', 40),
  (2, 'May', 30),
  (3, 'Tim', 22);
```

## Query SQL

```
SELECT * FROM `new_schema`.`users`;

SELECT * FROM `new_schema`.`orders`;
```

## Result

Query #1 **Execution time: 0ms**

| id | name | age |
|---|---|---|

| | | |
|---|---|---|
| 1 | John | 40 |
| 2 | May | 30 |
| 3 | Tim | 22 |

Query #2 **Execution time: 0ms**

| id | user_id | note |
|---|---|---|
| 1 | 1 | some information |
| 2 | 2 | some comments |
| 3 | 2 | no comments |
| 4 | 3 | more comments |

## Basic syntax & Usage

The setting of the foreign key needs to be done for the table, so the `ALTER TABLE` keywords need to be used as follows:

```
ALTER TABLE `new_schema`.`orders`
    ADD CONSTRAINT `orders_user_id_key`
    FOREIGN KEY (`user_id`)
    REFERENCES `new_schema`.`users` (`id`);
```

Here we can divide it into four parts: 1. `ALTER TABLE`: We have used this statement several times already; this statement specifies which table to set. 2. `ADD CONSTRAINT`: This comes before the name of the constraint you want to add. We set `orders_user_id_key` as the name of the constraint. 3. `FOREIGN KEY`: Refers to the foreign key you want to bind. We set `user_id` in this table as our foreign key. 4. `REFERENCES`: Specify the external data table to which the foreign key is bound. We set it to the `id` column in the `users` table.

## Protection from DELETE

After the previous statement has been executed, if we try to execute the delete statement to delete a user:

```
DELETE FROM `new_schema`.`users` WHERE (`id` = '1');
```

If we try to execute this command, an error message like this will appear to prevent us from deleting the user:

```
ERROR 1451: 1451: Cannot delete or update a parent row: a foreign key constraint fails
```

The reason for this message is that the user, whose id is 1, has been bound to the `user_id` column of some records in the `orders` table. This execution triggers the protection mechanism, thus preventing a missing data problem.

## Protection from UPDATE

Another scenario, assuming that we do not remove the data, but update the foreign key of the original `users` data, will also trigger the constraint which has the mechanism to protect the data. E.g:

```
UPDATE `new_schema`.`users` SET `id` = '6' WHERE (`id` = '1');
```

We will get the same error message:

```
ERROR 1451: 1451: Cannot delete or update a parent row: a foreign key constraint fails
```

The reason for the error is that we want to modify the user whose id is 1 and update its id to 6, which will cause the corresponding order records to find the inappropriate user and make an error. So our constraint kicks in to save us.

However, if we execute the following statements, no execution error will be encountered:

```
UPDATE `new_schema`.`users` SET `name` = 'Tony' WHERE (`id` = '1');
UPDATE `new_schema`.`orders` SET `user_id` = '3' WHERE (`id` = '1');
```

The reason is that the SQL we executed does not violate the logical correctness of the constraint. We only modify the value in other columns like `name`, or update the relative `users` record to another like `Jack`.

### Remove a Constraint

Since we can create a constraint, we must also be able to remove a constraint. The syntax for removing is as follows:

```
ALTER TABLE `new_schema`.`orders`
    DROP FOREIGN KEY `orders_user_id_key`;
```

It is worth mentioning that if you want to update a constraint, you need to remove the constraint first and then add the new constraint. There is no update relative statement.

## Diverse Constraints

In SQL, there are various modes for constraints, and we can set constraints individually for `UPDATE` and `DELETE` actions.

In addition to the settings for different behaviors, you can also set modes. As in the previous example, if you don't specify the mode when creating a constraint, the default mode is the NO ACTION.

NO ACTION is equivalent to RESTRICT mode in MySQL. As we observed, this mode protects data by prohibiting `UPDATE` and `DELETE` from being implemented when the command disobeys the constraint. If you are familiar with other SQL database systems, we recommend checking the details of the relevant official documents! But usually, these functions are similar for each database system.

## Syntax

In the syntax part, you can set the modes of different behaviors as follows:

```
ALTER TABLE `new_schema`.`orders`
```

```
    ADD CONSTRAINT `orders_user_id_key`
    FOREIGN KEY (`user_id`)
    REFERENCES `new_schema`.`users` (`id`)
    ON DELETE NO ACTION
    ON UPDATE RESTRICT;
```

You can focus on the two new phrases `ON DELETE` and `ON UPDATE`. With these two phrases, the mode can be set for `DELETE` and `UPDATE` behavior, and the name of the mode can be added directly after the phrases.

## Referential Actions

In the constraint, the official name for `NO ACTION` & `RESTRICT` is called Referential Actions, and these actions are commonly used. In addition to this, SQL usually provides the following two other modes:

1. `CASCADE`: Dynamically adjust the foreign key of another table when the constraint is triggered. For example, if the user's id is changed to 6, the `user_id` of the corresponding order records will also be updated to 6 together.
2. `SET NULL`: You can control to dynamically set the foreign key of another table to null when the behavior occurs. For example, if the user's id is changed to 6, the `user_id` corresponding to the order records will be changed to null.

For more details on the use of referential actions, it is recommended that you refer to MySQL Manual - FOREIGN KEY Constraints. Let's find out what kind of changes will be made!