Foreign Keys & Creating Tables

by Sophia

WHAT'S COVERED

This tutorial explores the use of foreign keys in more detail and the steps needed to ensure the keys are in the right tables in two parts:

1. One-to-one
2. One-to-many

1. One-to-one

As we have seen in prior tutorials, foreign keys are used to establish relationships between tables. Typically, we use a primary key in one table and a foreign key in another table to create a one-to-one or a one-to-many relationship between those two tables.

In a one-to-one relationship, one table serves as a parent table and the other table serves as a child table. With a foreign key constraint, a record must exist in the parent table before a related record can be added to the child table. In other words, a record in the child table must have a related record in the parent table.

An example of this could be an Employee table and a Benefits table. In the employee table, we would have information specific to the employee, such as their name, address, position, and date hired. In the benefits table, you could have investment plan, medical plan, dental plan, and life insurance plan information. In this scenario, it would not make sense to have the benefits data entered first, as it is dependent on the employee. It would be illogical to have a record in the benefits table that is not related to a record in the employee table.

The foreign key would be placed in the benefits table to reference the primary key of the employee table. We would take the primary key in the parent table, which in this case is employee_id, and copy it to use as a foreign key in the benefits table. This is where the term foreign key comes from, as the child table would have a primary key of its own, but the primary key that we are introducing from the parent table (Employee) is foreign to the child table (Benefit).

We can also see this one-to-one relationship in the set of representative and department tables that we previously created:

```
CREATE TABLE representative ( representative_id INT PRIMARY KEY, first_name VARCHAR (30) NOT NULL, last_name VARCHAR (30) NOT NULL );
```

```
CREATE TABLE department ( department_id INT PRIMARY KEY, department_name VARCHAR (100) NOT NULL, manager_id INT, constraint fk_manager FOREIGN KEY (manager_id) REFERE
);
```

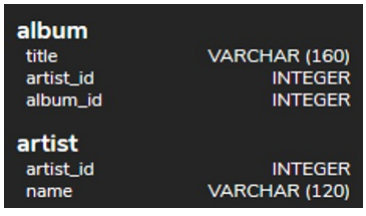
A single manager (representative) is associated with only one department, and a single department is associated with only one manager. Interestingly, this is one case where you could make the argument that either the department table or the representative table could be the parent table, because both representatives and departments can exist without the other in place. For example, we could have a department that recently lost its manager. It does not mean that we must remove the department, because it still exists, even though the manager is gone. Likewise, if we created a new department and do not have a manager for it yet, it is still able to exist. So you could make the argument that the following relationship is also valid:

```
CREATE TABLE department ( department_id INT PRIMARY KEY, department_name VARCHAR (100) NOT NULL );
```

```
CREATE TABLE representative ( representative_id INT PRIMARY KEY, first_name VARCHAR (30) NOT NULL, last_name VARCHAR (30) NOT NULL, department_id INT, constraint fk_ departı
```

2. One-to-Many

The one-to-many relationship is similar to the one-to-one relationship when it comes to foreign keys. However, it is clearer which table is the parent table and which table is the child table. Let's take a look at the Artist and Album table in our database. The one-to-many relationship follows the same guideline where we take a copy of the primary key from the table on the “one” side of the relationship (the parent table). We then incorporate it into the table on the “many” side (the child table). In this example, the artist_id that is the primary key in the Artist table becomes the foreign key in the Album table.



For every one-to-many relationship, the primary key of the “one” table will be a foreign key in the “many” table. It should never be the case that we have a foreign key on the “one” side of a one-to-many relationship.

Video Transcription

[MUSIC PLAYING] Foreign keys allow you to be able to ensure that you have data integrity by having data within one table referencing the data within another table. Typically this will be based on the primary care of the other table. Here's an example here. First of all, we have a representative table that we're going to create with the representative ID as being the primary key.

So now we'll go ahead and create the department table. In this case here, we'll have the department table that has the department ID as a primary key, the department name, and then the manager ID. Then afterwards, we define the constraint with the key word constraint, then the name of the constraint. In this case here, we'll just name it as FK_manager, then the keywords foreign key. And then we'll identify that we're going to be utilizing the manager ID, which is this one here. And it's going to reference the representative table with the representative ID within that representative table. This will enforce that any value that we place in for the manager ID has to exist in the representative table within that representative ID.

[MUSIC PLAYING]

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

SUMMARY

Foreign keys should be applied in the child table or the “many” side of a relationship.

Source: Authored by Vincent Tran