Foreign Keys & Creating Tables

by Sophia

WHAT'S COVERED

This lesson 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. Specifically, this lesson will cover:

1. One-to-One Relationships

A **foreign key** is a column, or a set of columns, that refers to the primary key of another table. As we have seen in prior lessons, 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, we could have an investment plan, medical plan, dental plan, and life insurance plan information. In this scenario, it would not make sense to enter the benefits data first, as it depends on the employee. It would be illogical to have a record in the benefits table that is unrelated 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 is employee_id, and copy it 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 FOR
);
```

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 create a new department and still need a manager for it, it is still able to exist. So, you could make the argument that the following relationship is also valid:

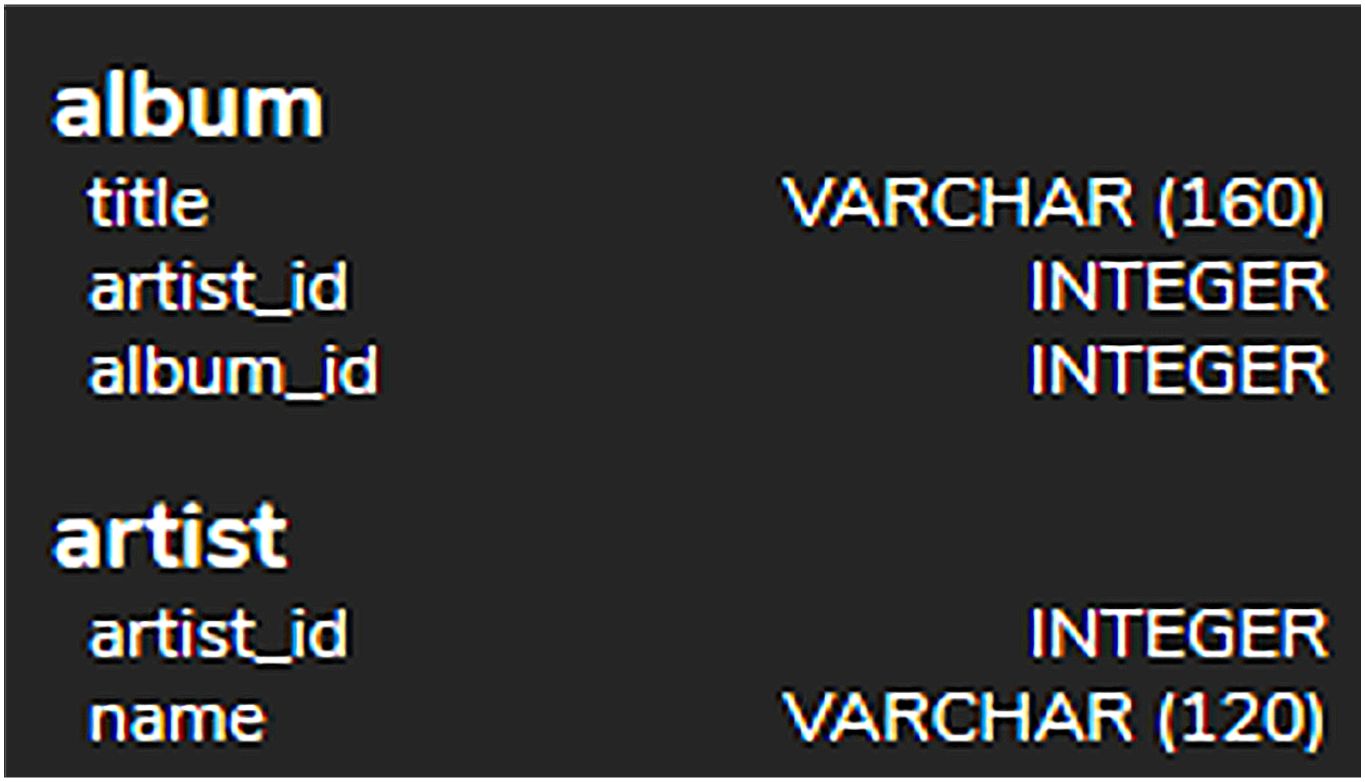
```
CREATE TABLE department (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL
);

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_department FOREIGN KEY
  (department_id) REFERENCES department(department_id)
);
```

TERM TO KNOW

Foreign Key
A column or set of columns that refers to the primary key of another table.

2. One-to-Many Relationships

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) and 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.



```
album
  title                 VARCHAR (160)
  artist_id                   INTEGER
  album_id                    INTEGER

artist
  artist_id                   INTEGER
  name                  VARCHAR (120)
```

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.

WATCH

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

In this lesson, you learned that foreign keys play a pivotal role in maintaining database relational integrity because they establish and enforce relationships between tables. A foreign key is a column or set of columns that refers to the primary key of another table.

You learned that in a **one-to-one relationship**, each of the two tables involved uses the same information for its primary key, so the relationship can be set up so that either table is the primary key and the other the foreign key. That's because in a one-to-one relationship, determining which table is the parent and which is the child is situational. In contrast, in a **one-to-many relationship**, the "one" side is always the primary key, and the "many" side is always the foreign key. You also learned how to set up a table as the foreign key of another table during the table's creation. You will learn more about creating foreign keys in upcoming lessons.

TERMS TO KNOW

Foreign Key

A column or set of columns that refers to the primary key of another table.