Table Constraints

by Sophia

WHAT'S COVERED

In this lesson, you will explore the different table constraints that can be applied, in four parts. Specifically, this lesson will cover:

1. Constraints and Referential Integrity

Constraints are rules that you can apply to tables and their fields (attributes) to place restrictions on what can be entered in them. Constraints help ensure that the data entered in a table is accurate and reliable. For example, one common constraint is to set a certain field to require a unique entry for each record. When a data action would violate a constraint, the database system does not allow it.

There are two kinds of constraints: column-level and table-level. Column-level constraints apply to a specific column (field), and table-level constraints apply to the table as a whole.

Constraints exist in order to enforce referential integrity. **Referential integrity** refers to the ability of a database to prohibit operations that might compromise the database's integrity.

These are the main rules of referential integrity:

- Primary key constraints: Each primary key in a table must be unique and without null values.
- Foreign key constraints: A foreign key in a table must match up with an existing primary key value in another table.
- Cascading operations: When a record is changed or deleted from a table on the 1 side of a relationship, there must be a plan for what to do with the corresponding records on the many side of the relationship. For example, if a customer's customerID changes, should the customerID change in each of their orders in the order table?

The following constraints are used in SQL:

- NOT NULL—Ensures that a column cannot have a NULL value.
- UNIQUE—Ensures that all values in a column are different.
- PRIMARY KEY—Designates the primary key field in the table, which applies the *UNIQUE and NOT NULL constraints and indexes the field.
- FOREIGN KEY—Prevents actions that would destroy links between tables.
- CHECK—Ensures that the values in a column satisfy a specific condition.
- DEFAULT—Sets a default value for a column if no value is specified.
- CREATE INDEX—Indexes the specified column so that searches based on it run quickly.

The rest of this lesson explains the specifics of applying constraints.

TERMS TO KNOW

Constraint
A rule that imposes limitations on what can be done. In the context of a database table, a constraint limits what data can be entered.

Referential Integrity
A set of constraints that ensure data and relationship integrity by enforcing rules about primary key constraints, foreign keys, and cascading operations.

2. The PRIMARY KEY, NOT NULL, and UNIQUE Constraints

In a previous lesson, we made use of the PRIMARY KEY constraint but didn't really explain what it was used for.

The PRIMARY KEY constraint is a special type of index that also applies the NOT NULL and UNIQUE constraints. There can be only one primary key per table. The primary key serves as a way to uniquely identify each record.

In order to understand the PRIMARY KEY constraint, you first need to understand the NOT NULL and UNIQUE constraints.

The NOT NULL constraint is used to ensure that the field contains a value in every record. It prevents an important field from being left blank. For example, in the artist table in the PostgreSQL database we have been working with, the artist's name field should be NOT NULL because that field is essential in identifying the artist. When you apply the PRIMARY KEY constraint, NOT NULL is applied automatically, but you can also apply NOT NULL separately to any field(s) in the table.

In the employee table, many of the columns could be set up to be required values using the NOT NULL constraint. For example, the hire_date would be one value that should always exist, as the hire_date would need to exist for any employee hired into the company. To add a NOT NULL constraint to a column, we would list it beside the data type like this:

```
CREATE TABLE contact (
contact_id int PRIMARY KEY,
username VARCHAR(50) NOT NULL,
password VARCHAR(50) NOT NULL
);
```

The NOT NULL constraint in this example will not permit the username and password to have missing values in the table when trying to insert the contact_id.

The other constraint that the PRIMARY KEY constraint uses is the UNIQUE constraint. Like the NOT NULL constraint, the UNIQUE constraint can be set up on its own. The UNIQUE constraint requires each record to have a different value for that field. The exception is that an attribute in the column could be empty. For example:

```
CREATE TABLE newsletter  email VARCHAR(50) UNIQUE );
```

In the example above, the email must be unique, but it potentially permits no value to be inserted. This could be a problem because it is the only attribute in the table.

PRIMARY KEY combines NOT NULL and UNIQUE, so not only can the field not be empty, but each entry must be unique. This enables the primary key to uniquely identify any record within the table. In the same example of the newsletter, having the email be the primary key would ensure we have all the values.

```
CREATE TABLE newsletter (email VARCHAR(50) PRIMARY KEY );
```

KEY CONCEPT

A primary key does not have to be for a single column. It can combine multiple columns together as a special type of primary key called a composite key, which you learned about earlier in the course. A composite key takes the combination of two or more columns together to identify a row within a table uniquely.

3. The FOREIGN KEY Constraint

The FOREIGN KEY constraint is an important one to help maintain referential integrity. It is used to link two tables together. The **foreign key** in one table refers to the primary key in another table. For example, suppose you want to create a table for pet owners that contains three columns: owner_id, firstname, and lastname. Here's the statement for doing that:

```
CREATE TABLE owner (
owner_id int PRIMARY KEY,
firstname VARCHAR(40) NOT NULL,
lastname VARCHAR(40) NOT NULL
);
```

Then you want to create a pet table that will be connected to the owner table by the owner_id field. That field is the primary key in the owner table (shown above) and the foreign key in the pet table. Here's the code you would use for that.

```
CREATE TABLE pet (
pet_id int PRIMARY KEY,
pet_name VARCHAR(40) NOT NULL,
species VARCHAR(40) NOT NULL,
breed VARCHAR(40) NULL,
Birthdate DATE,
owner_id int,
CONSTRAINT fk_owner
FOREIGN KEY (owner_id)
REFERENCES owner(owner_id)
);
```

Let's break down what is happening in the statement for creating the pet table. The owner_id field in the pet table requires a constraint in order to create the relationship between it and the owner_id field in the owner table.

CONSTRAINT fk_owner defines which table is being connected to; fk stands for foreign key.

FOREIGN KEY (owner_id) defines which field in the pet table should be on the “many” side of the relationship. You know it's the main side because it's the foreign key; recall that the primary key side is the “one” side.

REFERENCES owner(owner_id) defines which field in the owner table should be on the “one” side of the relationship.

This constraint will prevent a record from being deleted from the owner table if there is at least one row in the pet table that references that owner. It will also prevent adding a record in the pet table where the owner_id value in the pet table does not match an owner_id in the owner table. If you tried to do either of those actions, you would get an error message.

Try it for yourself. After creating the tables using the above code, run the following statement:

```
INSERT INTO owner (owner_id, lastname, firstname)
VALUES (1, 'Smith', 'John');
```

You will learn later in the course how to insert records; for now, just copy that code.

Now we will try to create a record in the pet table using an owner_id that doesn't exist:

```
INSERT INTO pet (pet_id, owner_id, birthdate, pet_name, species)
VALUES (1, 2, '2023-04-05', 'Sparky', 'Canine');
```

You'll see this error message:

Query failed because of: error: insert or update on table “pet” violates foreign key constraint “fk_owner”

Now, in the VALUES clause, change the 2 to a 1 in the second position, and rerun the statement:

```
INSERT INTO pet (pet_id, owner_id, birthdate, pet_name, species)
VALUES (1, 1, '2023-04-05', 'Sparky', 'Canine');
```

This time it worked! To make sure, use a SELECT statement to see the records in the pet table:

```
SELECT * FROM pet;
```

TERM TO KNOW

Foreign Key
In a one-to-many relationship, the related field in the table on the “many” side.

4. The DEFAULT and CHECK Constraints

Next, let's look at two more commonly used constraints.

The DEFAULT constraint assigns a value to an attribute whenever a new row is added to a table if a value is not set for it. This can be useful to set a base value for an attribute. For example, in a table that tracks registrations for an event, you might define the registration_date field like this:

```
registration_date DATE DEFAULT CURRENT_DATE
```

The CHECK constraint can be used to validate data when an attribute is entered. For example, we could do checks of items such as:

- Check that the unit_price in the track table has a value of >=0, as there should be no negative price.
- Check that the hire_date in the employee table is greater than January 01, 2000, as that was the date that the company opened.
- Check that the customer's email has a standard email format.

To continue with the example of a registration table, you might restrict the age field to ages between 18 and 100, to make sure only adults register:

```
age CHECK (age >=18 AND age <=100)
```

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 about several **constraints** that limit entries in table columns, noting that constraints exist in order to enforce **referential integrity.** You learned that the **PRIMARY KEY constraint** is a combination of **UNIQUE and NOT NULL constraints**, and that the **FOREIGN KEY constraint** prevents entries that would invalidate a link between tables. The **CHECK constraint** applies conditions to limit what can be entered, and the **DEFAULT constraint** sets a default value for the column if no value is entered. CREATE INDEX indexes the specified column. All of these constraints can help ensure that columns will accept only values that are valid for your business rules.

TERMS TO KNOW

Constraint
A rule that imposes limitations on what can be done. In the context of a database table, a constraint limits what data can be entered.

Foreign Key
The term foreign key refers to when a primary key from one table appears in another.

Referential Integrity
A set of constraints that ensure data and relationship integrity by enforcing rules about primary key constraints, foreign keys, and cascading operations.