CREATE TABLE Syntax

by Sophia

WHAT'S COVERED

In this lesson, you will explore using the CREATE TABLE statement to generate tables in the database, in two parts. Specifically, this lesson will cover:

1. Rules and Data Types

Every database system you will use has its own conventions for creating and modifying tables and their attributes. It's important for a database designer to check the system's documentation to learn about its rules and restrictions. Some common rules are consistent across many different database systems. These include:

- The table and column names must start with a letter.
- The table and column names can only contain letters, numbers, and underscores.
- The table and column names should not contain spaces (though some databases may allow this).
- Each system has a maximum length for the names of tables and columns. For example, a PostgreSQL column name can be up to 59 characters, while a table name can be up to 63 characters.

2. Data Types

In other lessons, we’ve called out specific data types, like VARCHAR for variable characters (numbers and letters for addresses) and INT for whole numbers. In most systems, many data types are available for holding numbers, text, dates, and more.

When designing a database, it's important to consider what kinds of data each table will contain and what the optimal data type should be for each column. For example, a table might have a State column. You must decide whether it will hold full state names or their two-letter abbreviations because it makes a difference in the maximum number of characters to specify.

For each column or field in a table, we must identify the type of data the column can store, along with its length, in some cases. The most common data types are:

- Boolean: Stores true, false, or null (no value).
- CHAR(n): A fixed-length character of length n with space added. If an entry is shorter than the fixed length, PostgreSQL pads extra spaces up to the length (n) of the column. It generates an error if you try to insert a longer value than that column's maximum.
- VARCHAR(n): A variable-length character string storing up to n characters. Note that with VARCHAR, unlike CHAR, PostgreSQL does not pad the spaces when the stored string is shorter than the length of the column.
- TEXT: A variable-length character string that has unlimited length.
- SMALLINT: A small integer is a 2-byte signed integer ranging from -32,768 to 32,767.
- INT: An integer is a 4-byte integer ranging from -2,147,483,648 to 2,147,483,647.
- SERIAL: An automatically generated integer, often used to assign unique primary key values such as OrderID. We'll cover this in an upcoming lesson.
- Float(n): A floating point number (that is, a number where the decimal point can be in any position). The (n) number specifies a total number of digits that can be stored in the number, including the digits before and after the decimal point. The data type does not have a fixed number of bytes.
- Real: A 4-byte floating point number. It's suitable for storing values that have decimal points but not a lot of decimal places.
- Numeric(P,S): A real number with P digits, with S number of digits that are displayed after the decimal point.
- Date: Stores dates without times.
- Time: Stores times without dates.
- Timestamp: Stores both dates and times.

Each of these data types have their own default values that vary between systems.

3. Using the CREATE TABLE Command

The SQL CREATE TABLE command makes a new table by defining the layout of the table. Similar to the SELECT statement you've learned about, there is a standard structure for the format of the CREATE TABLE statement. The CREATE TABLE statement includes the name of the table and the names, data types, and lengths of the columns. Here's an example that creates a table called customers:

```
CREATE TABLE customers (
  customerID INT NOT NULL AUTO_INCREMENT,
  emailAddress VARCHAR(255) NOT NULL,
  password VARCHAR(60) NOT NULL,
  firstName VARCHAR(60) NOT NULL,
  lastName VARCHAR(60) NOT NULL,
  shipAddressID INT DEFAULT NULL,
  billingAddressID INT DEFAULT NULL,
  PRIMARY KEY (customerID),
  UNIQUE INDEX emailAddress (emailAddress)
);
```

The customerID is a whole number (INT) that must have data in it (NOT NULL) and must auto-increment. As you can see in the next-to-last clause, customerID will be the primary key field.

Notice that the last clause creates a prebuilt index of the emailAddress field. That will make it quicker to look up a customer's ID number from their email address. A unique index on a field indexes each record's entry in that field as it is entered; this makes SELECT statements run much more quickly when they reference the indexed field in a WHERE clause.

Email, password, and first and last time are all variable characters (VARCHAR) and must have data in them when they are inserted into the database.

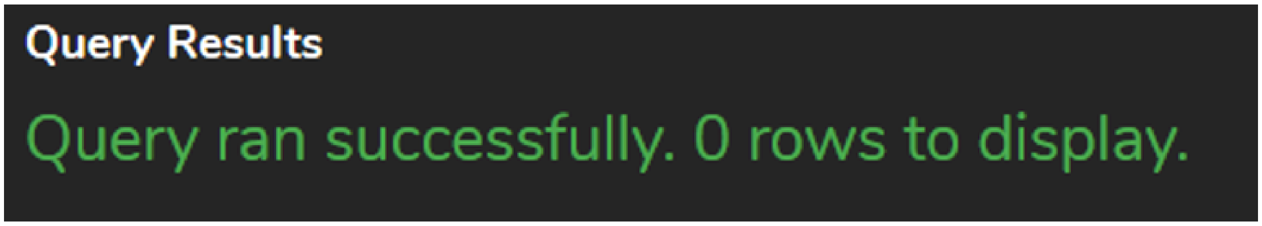
Shipping address and billing address are ID numbers (INT), that are default NULL, meaning entering data in these fields is optional.

To make the code easier to read, having one line per column or attribute is good practice.

If we wanted to create a new table called contact that had the contact_id as the primary key and two additional columns with the username and password, the statement could look like the following:

```
CREATE TABLE contact (
contact_id INT PRIMARY KEY,
username VARCHAR(50),
password VARCHAR(50)
);
```

Running the above statement in our sample PostgreSQL database, you should see a message similar to the following:

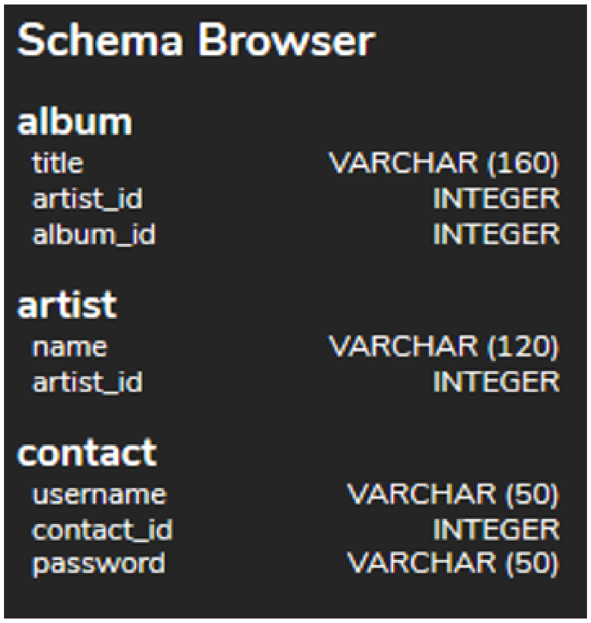


Query Results

Query ran successfully. 0 rows to display.

Note: When you see the query ran successfully, with zero rows to display, that means it actually did something. There are zero rows because the new table does not yet contain any data.

After you run the above statement to create the contact table, you should see it listed after the album and artist tables in the schema browser.



Schema Browser

album	
title	VARCHAR (160)
artist_id	INTEGER
album_id	INTEGER

artist	
name	VARCHAR (120)
artist_id	INTEGER

contact	
username	VARCHAR (50)
contact_id	INTEGER
password	VARCHAR (50)

A table could have just one column or many columns, depending on what you are trying to create. For example, you could have a newsletter table that just consists of emails:

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

Although this is the basic syntax, there are other CREATE TABLE criteria that we'll explore in upcoming lessons.

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 **rules and data types**. You learned to use the **CREATE TABLE** command in PostgreSQL to create a new table and define its fields. You learned about the various data types available and the required naming and sizing conventions. You also learned how to make a field required (NOT NULL) or optional (NULL). Lastly, you practiced creating a new table that included a primary key designation, required and optional fields, and a unique index.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND Faithe Wempen (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR **TERMS OF USE**.