# INSERT INTO to Add Row

*by Sophia*

WHAT'S COVERED

In this lesson, you will use INSERT statements to add a single row into an existing table, in two parts. Specifically, this lesson will cover:

1. Using the INSERT INTO Statement
2. Examples

## 1. Using the INSERT INTO Statement

We have used the **INSERT INTO** statement a few times in prior lessons, but we will get a chance to explore the details around the statement better in this lesson. Remember that the basic syntax of the INSERT statement looks like this:

```
INSERT INTO <tablename> (<column1>, <column2>, ...)
VALUES (<value1>, <value2>, ...);
```

This statement first identifies the table in which to insert the new record. Then, in parentheses, it lists the column names, separated by commas.

The next clause, VALUES, is followed by another set of parentheses, in which the values for each of the columns listed in the previous clause are specified. Entries that are dates, characters, or strings must be enclosed in single quotes. This operation must include all required columns (that is, all columns with a PRIMARY KEY or NOT NULL constraint). The only exception to that is if the column has a DEFAULT value or a sequence attached to it, which you will learn about in the next lesson.

For example, if we had a table named links where we wanted to collect interesting links for us to use later, we could use a webform to insert that link. The underlying INSERT INTO would look like this:

```
INSERT INTO links (url, name)
VALUES('https://www.sophia.org,'PostgreSQL lesson');
```

In the above example, we are inserting into the links table, in two columns: url and name. For the url, we are inserting https://www.sophia.org. It is enclosed in single quotes because it is a string. For the name column, we are inserting PostgreSQL lesson.

TERM TO KNOW

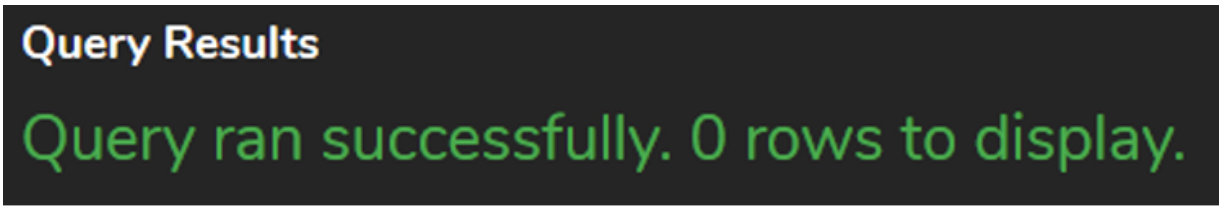
**INSERT INTO**

A statement that creates a new row or updates an existing row in a table.

# 2. Examples

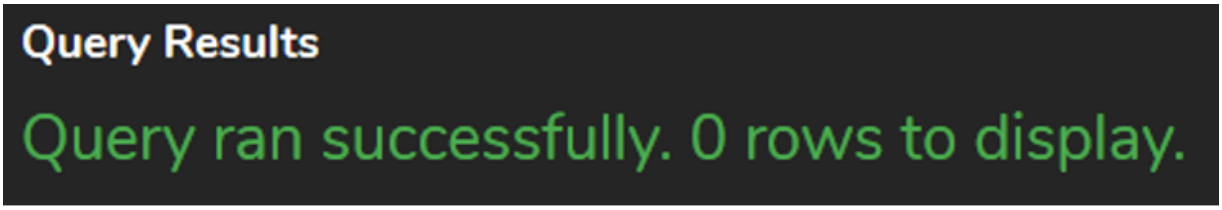
For example, to insert into the artist table, we can run the following statement:

```
INSERT INTO artist (artist_id, name)
VALUES (1000,'Bob Dylan');
```



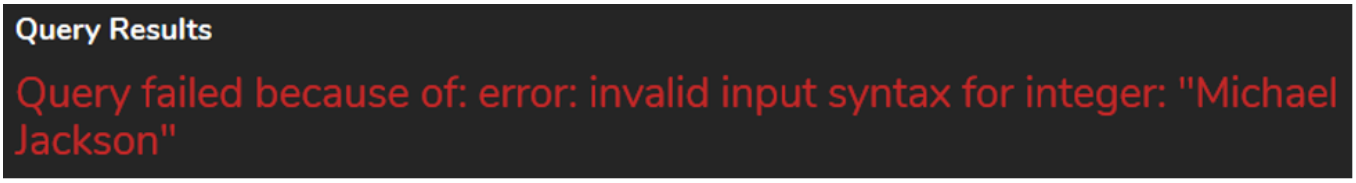

The column order in the statement does not matter; they don't have to be in the same order in which they appear in the table. For example, the following statement reverses the order of the columns but still works equally well.

```
INSERT INTO artist (name, artist_id)
VALUES ('Michael Jackson', 1001);
```




However, we must make sure that the order of the columns in the VALUES clause matches the order in the INSERT INTO clause. Otherwise, the data will be placed in the wrong column. You might or might not see an error message if you make that mistake; it depends on whether the data is erroneously being entered into a column with a data type that permits the data being entered. In the example below, it does not match, resulting in an error.

```
INSERT INTO artist (name, artist_id)
VALUES (1001, 'Michael Jackson');
```




Some programmers try to use this as a way to do error checking on the data that is being inserted, but overall, this is an unreliable method for error checking. It catches that we tried to insert a string into an integer column—but only that issue. There are many other instances where we may insert wrong data into the same data type, like swapping two integer value columns. It is important that we double-check the order of the column list and the value list to make sure everything matches, and the right data goes into the right column.

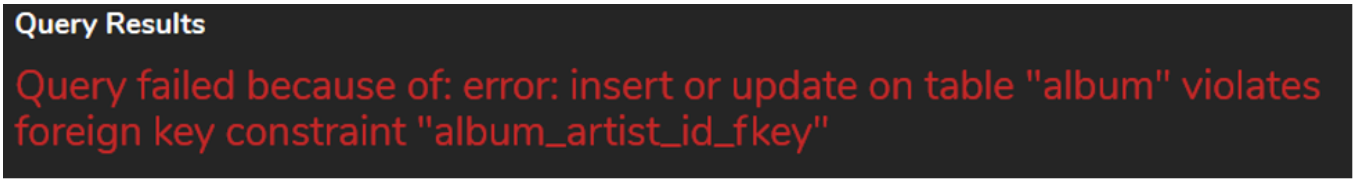
We should also be aware that there might be existing constraints on a table, such as foreign keys, as this can prevent data from being inserted. For example, consider if we tried to insert the following statement:

```
INSERT INTO album (artist_id, album_id, title)
VALUES (999, 2000,'Latest Hits');
```

The artist_id in the album table references the artist table. If we looked at the artist table for the artist_id equal to 999, we would find that this doesn't exist:

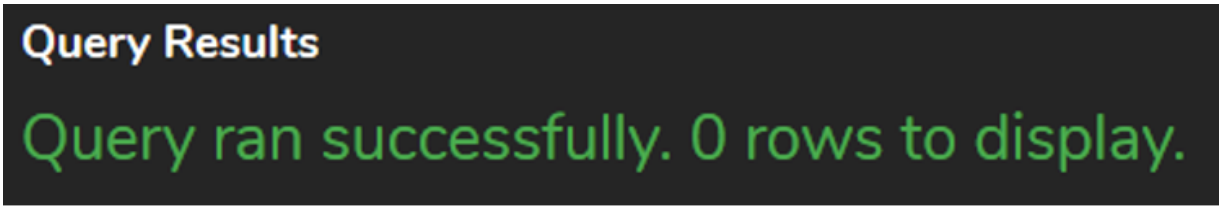
```
SELECT *
FROM artist
WHERE artist_id = 999;
```

As such, trying to insert it into the table would result in an error:



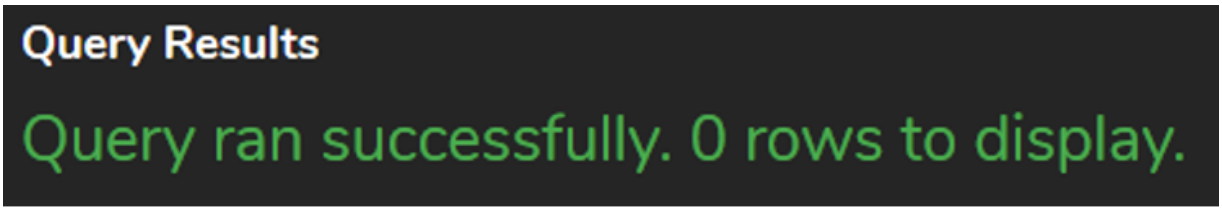

If we wanted to add the album *Latest Hits* to the album table using the artist_id 1000 that we inserted for Bob Dylan, the INSERT statement would look like this:

```
INSERT INTO album (artist_id, album_id, title)
VALUES (1000, 1001,'Latest Hits');
```




However, consider if we accidentally swapped the values, as follows:

```
INSERT INTO album (artist_id, album_id, title)
VALUES (1001, 1000,'Latest Hits');
```




No error would be displayed in this case, as we also have added an artist with the artist_id value of 1001. This is why it is important to be careful regarding what the values represent, as the database will not catch these logical errors.

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 **using the INSERT INTO statement** in PostgreSQL adds new rows of data into a table. You can insert data into specific columns or the entire row by specifying the table name and providing values for the corresponding columns. The VALUES clause lets the user specify the values for each column. You also learned by looking at some **examples** that the columns can be referenced in any order but must be referenced in the same order in both the INSERT INTO clause and the VALUES clause.

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

## TERMS TO KNOW

**INSERT INTO**
A statement that creates a new row or updates an existing row in a table.