DROP TABLE to Remove Tables

by Sophia

WHAT'S COVERED

In this lesson, you will use the DROP TABLE statement to remove a table, in three parts. Specifically, this lesson will cover:

1. Removing a Table With DROP TABLE

You can remove a table from a database using the **DROP TABLE** statement. Beware, though: A dropped table is instantaneously gone for good. Unless you have a current backup of it, its data is lost forever. The DROP TABLE statement also drops the indexes, constraints, synonyms, triggers, and access privileges associated with it.

After you drop a table, you will also need to remove all the associated views, dashboards, and programs others have made that work with that table and all the diagnostic data. So, dropping a table should never be done lightly.

For DROP TABLE, the structure of the statement looks like the following:

```
DROP TABLE <tablename>;
```

DROP TABLE is one of the easier statements to write, but it can get a bit more complex when there are relationships between tables. Any relationship with one or more foreign keys cannot be dropped until those relationships are removed from the database.

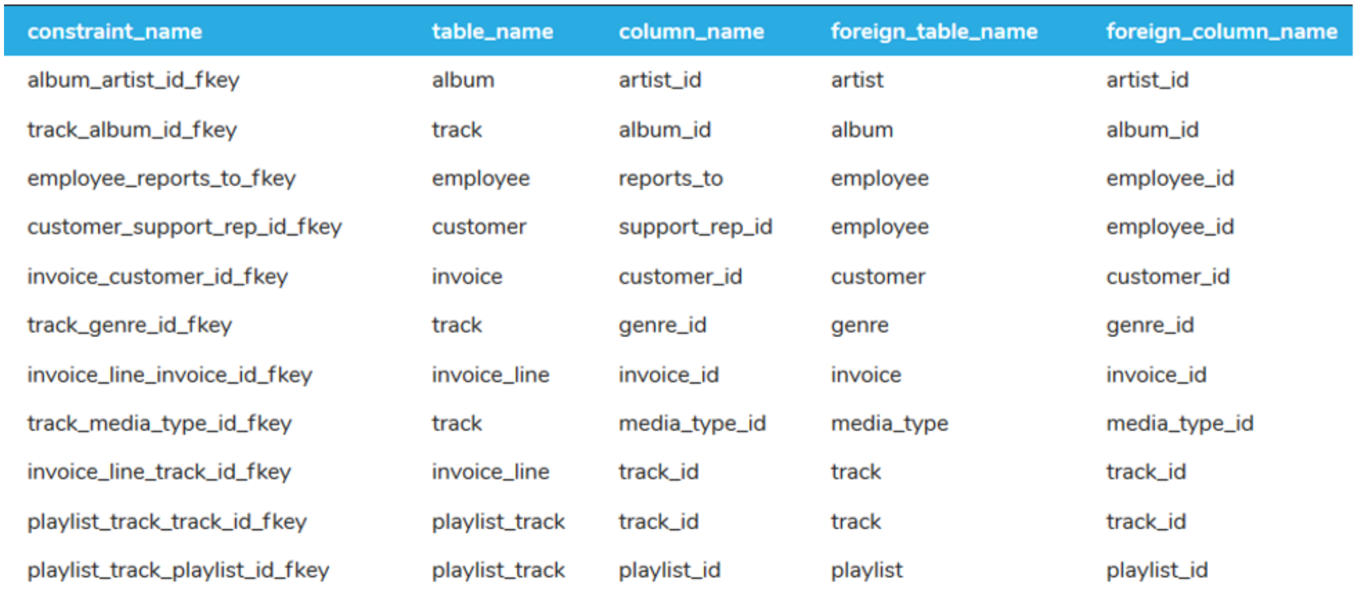
In a relationship between tables (foreign keys), you can only drop a table if it is not the “one” in a one-to-many relationship. We will get into more specifics about table relationships later in the course, but it is important to be aware of this detail here. If we try to drop a table with an existing relationship with other tables, we will get an error. The order in which we drop tables depends on the foreign key constraints.

TERM TO KNOW

DROP TABLE
A SQL statement that removes a table from a database.

2. Relationship Considerations When Dropping Tables

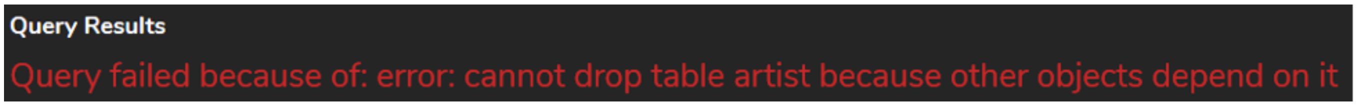
The sample database we've been working with in PostgreSQL has the following foreign keys set up across the various tables:



constraint_name	table_name	column_name	foreign_table_name	foreign_column_name
album_artist_id_fkey	album	artist_id	artist	artist_id
track_album_id_fkey	track	album_id	album	album_id
employee_reports_to_fkey	employee	reports_to	employee	employee_id
customer_support_rep_id_fkey	customer	support_rep_id	employee	employee_id
invoice_customer_id_fkey	invoice	customer_id	customer	customer_id
track_genre_id_fkey	track	genre_id	genre	genre_id
invoice_line_invoice_id_fkey	invoice_line	invoice_id	invoice	invoice_id
track_media_type_id_fkey	track	media_type_id	media_type	media_type_id
invoice_line_track_id_fkey	invoice_line	track_id	track	track_id
playlist_track_track_id_fkey	playlist_track	track_id	track	track_id
playlist_track_playlist_id_fkey	playlist_track	playlist_id	playlist	playlist_id

Notice that the album table has the artist_id as a foreign key to the artist table's artist_id. If we tried to run a DROP TABLE on the artist table, we should get the following error:

```
DROP TABLE artist;
```



```
Query Results
Query failed because of: error: cannot drop table artist because other objects depend on it
```

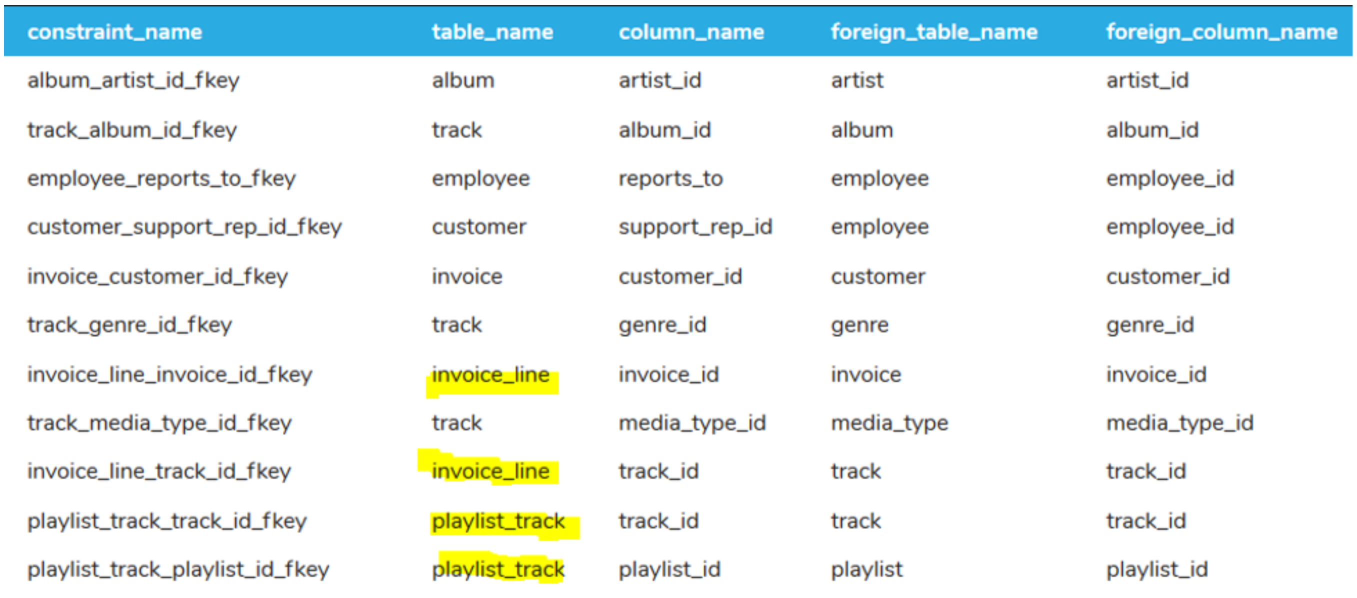
Because a table has relationships with other tables, you need to find the constraints first (in this case, the foreign key) and remove them. Then you can work out the order in which to drop tables in the database. To do this, we need to pull up all the constraints. You can do that by running this command:

```
SELECT conrelid::regclass AS table_name,
   conname AS foreign_key,
   pg_get_constraintdef(oid)
FROM pg_constraint
WHERE contype = 'f'
```

```
AND connamespace = 'public'::regnamespace
ORDER BY conrelid::regclass::text, contype DESC;
```

Keep this command handy, as it is a generic way of listing out all the foreign keys in a database for every table. This presents those foreign keys in a table format that you can export and use as documentation later.

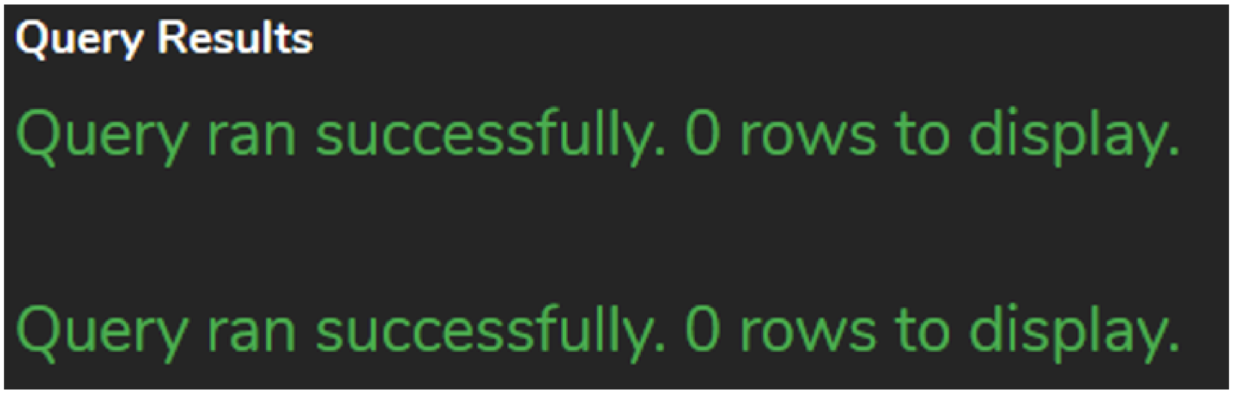
As shown below, the two tables that do not have a foreign key are the invoice_line table and the playlist_track table.



constraint_name	table_name	column_name	foreign_table_name	foreign_column_name
album_artist_id_fkey	album	artist_id	artist	artist_id
track_album_id_fkey	track	album_id	album	album_id
employee_reports_to_fkey	employee	reports_to	employee	employee_id
customer_support_rep_id_fkey	customer	support_rep_id	employee	employee_id
invoice_customer_id_fkey	invoice	customer_id	customer	customer_id
track_genre_id_fkey	track	genre_id	genre	genre_id
invoice_line_invoice_id_fkey	invoice_line	invoice_id	invoice	invoice_id
track_media_type_id_fkey	track	media_type_id	media_type	media_type_id
invoice_line_track_id_fkey	invoice_line	track_id	track	track_id
playlist_track_track_id_fkey	playlist_track	track_id	track	track_id
playlist_track_playlist_id_fkey	playlist_track	playlist_id	playlist	playlist_id

We should be able to drop them both without any errors. Notice that we have a semicolon at the end of each line; these are separate commands that are run sequentially. As such, the query results will include a successful result message for each individual statement.

```
DROP TABLE invoice_line;
DROP TABLE playlist_track;
```



```
Query Results

Query ran successfully. 0 rows to display.

Query ran successfully. 0 rows to display.
```

Next, we can look at the initially referenced tables to see if they have any relationships that still remain as foreign keys:

constraint_name	table_name	column_name	foreign_table_name	foreign_column_name
album_artist_id_fkey	album	artist_id	artist	artist_id
track_album_id_fkey	track	album_id	album	album_id
employee_reports_to_fkey	employee	reports_to	employee	employee_id
customer_support_rep_id_fkey	customer	support_rep_id	employee	employee_id
invoice_customer_id_fkey	invoice	customer_id	customer	customer_id
track_genre_id_fkey	track	genre_id	genre	genre_id
invoice_line_invoice_id_fkey	invoice_line	invoice_id	invoice	invoice_id
track_media_type_id_fkey	track	media_type_id	media_type	media_type_id
invoice_line_track_id_fkey	invoice_line	track_id	track	track_id
playlist_track_track_id_fkey	playlist_track	track_id	track	track_id
playlist_track_playlist_id_fkey	playlist_track	playlist_id	playlist	playlist_id

Since they do not, we can go ahead and drop those tables as well:

```
DROP TABLE invoice;
DROP TABLE track;
DROP TABLE playlist;
```

Then, looking at those tables, we can track down which tables referenced them:

album_artist_id_fkey	album	artist_id	artist	artist_id
customer_support_rep_id_fkey	customer	support_rep_id	employee	employee_id
employee_reports_to_fkey	employee	reports_to	employee	employee_id

Although we have the album, artist, customer, employee, genre, and media_type tables left, if we drop the album and customer tables, we can drop the rest, as the employee table is linked to itself. We can drop all of the tables by running the following:

```
DROP TABLE album;
DROP TABLE customer;
DROP TABLE employee;
DROP TABLE artist;
DROP TABLE genre;
DROP TABLE media_type;
```

That would remove all the tables in the database. Normally you wouldn't do that. If you wanted to delete an entire database, an easier way to do it would be to back up the database and then delete the database as a whole, rather than dealing with the individual tables. However, there may be a situation where you need to keep a database's parameters but recreate all the tables.

HINT

When you delete a table, you will also need to delete all the views, dashboards, and other programs that reference that table. Otherwise, people may try to use them, and errors will appear. You will want to go back and verify that all associated objects have been backed up and then removed.

In review, we break down the order of the dropping of the tables into four separate sets of statements, starting with dropping the tables that had no foreign keys linked to them:

```
DROP TABLE invoice_line;
DROP TABLE playlist_track;
```

Then, we drop the tables that only had foreign keys to those tables that were dropped:

```
DROP TABLE invoice;
DROP TABLE track;
DROP TABLE playlist;
```

Next, we drop the tables that had foreign keys to those tables dropped:

```
DROP TABLE album;
DROP TABLE customer;
```

Lastly, we proceed with the rest of the tables, as they no longer had any foreign keys that held them back from being dropped:

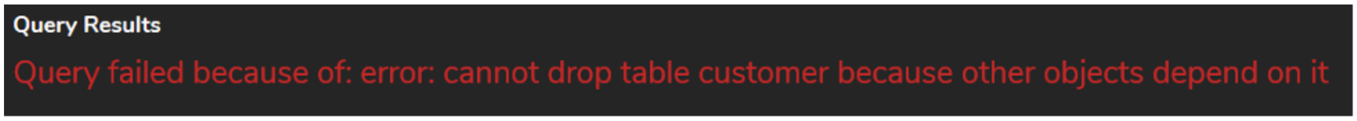
```
DROP TABLE employee;
DROP TABLE artist;
DROP TABLE genre;
DROP TABLE media_type;
```

3. Using the CASCADE and IF EXIST Options

In the previous section, you learned how to manually track down and remove the associated relationships and constraints that might prevent a table from being dropped. There may be times when you don't have time to track down all the foreign keys in a complex database, though. In these situations, you may find it easier to use the DROP TABLE statement's CASCADE option. This will drop the table and also remove any constraints that link to the table. For example, consider if we tried to drop the CUSTOMER table:

```
DROP TABLE customer;
```

We should get the following error, as the customer_id is used in the invoice table:


```
Query Results
Query failed because of: error: cannot drop table customer because other objects depend on it
```

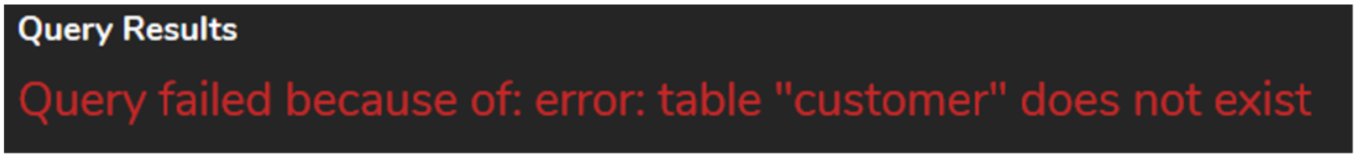
To avoid this error, you can instead run the command with the CASCADE option:

```
DROP TABLE customer CASCADE;
```

The CASCADE option drops the table and also removes the foreign key constraint on the invoice table for the customer_id.

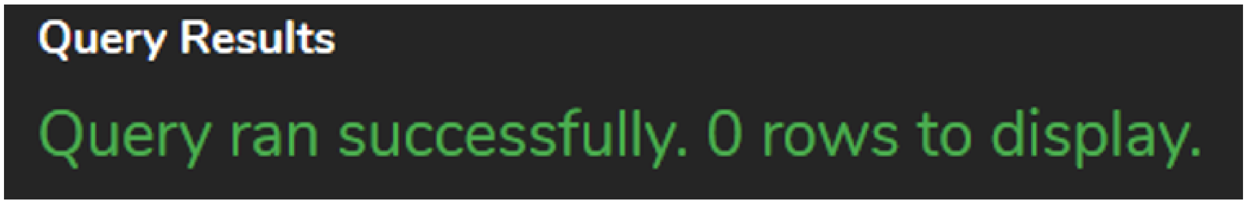
Another option with the DROP TABLE statement is the IF EXISTS option. If you try to drop a table that doesn’t exist or has already been dropped, you will get an error:

```
DROP TABLE customer;
```


```
Query Results
Query failed because of: error: table "customer" does not exist
```

However, adding the IF EXISTS will allow you to still run the command:

```
DROP TABLE IF EXISTS customer;
```


```
Query Results
Query ran successfully. 0 rows to display.
```

This can be useful when you include multiple commands together and do not want the database to stop on an error.

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

During this lesson, you learned that the **DROP TABLE** command in PostgreSQL permanently **removes an existing table** and all its associated data from the database. This tool can delete an entire table, including its schema, columns, indexes, and constraints. You learned that there are **relationship considerations when dropping tables**, requiring you to delete related tables and relationships in a certain order. You learned how to manually manage that order, and then you learned how to automate those considerations **using the CASCADE and IF EXISTS options**.

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

DROP TABLE
A SQL statement that removes a table from a database.